Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

SCHULICH School of Engineering
UNIVERSITY OF CALGARY

Degree Course Systems Engineering

Major: Infotronics

Diploma 2013

Franco Summermatter

Force Controlled Haptic Experiment With Two Different Robots

Professor
Joseph Moerschell

Expert
Chris Macnab

Submission date of the report
27.09.2013

Valais excellence

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2012/13**	No TD / *Nr. DA* **it/2013/25**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Franco Summermatter** Professeur / *Dozent* **Joseph Moerschell**	Lieu d'exécution / *Ausführungsort* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Chris Macnab | University of Calgary**	

Titre / *Titel*

Haptische Steuerung eines Roboterarms

Beschreibung

Eine haptische Steuerung besteht aus einem Meister und einem Sklaven, welche über elektrische Steuersignale verbunden sind. Der Sklave imitiert die Bewegungen des Meisters. Der Meister gibt dem Benutzer das Widerstandsmoment des Sklaven zurück.

In dieser Arbeit wird das Konzept auf einen 2-dimensionalen Roboterarm angewandt. Im Vordergrund steht hier die Anwendung eines Steuerungskonzeptes, welches die Kraft und nicht die Position regelt. Dieses Konzept der Steuerung soll in beide Richtungen Implementiert werden, d.h. Meister und Sklave vertauschen Ihre Rollen (2 unabhängige Anwendungen). Somit kann das Steuerungskonzept auf unterschiedlichen Hardwarebedingungen untersucht werden. Zur Realisierung wird Matlab/Simulink verwendet.

Ziele

— Inbetriebnahme der Hardware.
— Implementieren der haptischen Steuerung vom Phantom Omni (Master) zum 2DSFJ-Roboter (Slave)
— Implementieren der haptischen Steuerung vom 2DSFJ-Roboter (Master) zum Phantom Omni (Slave)

Signature ou visa / *Unterschrift oder Visum* Resp. de la filière *Leiter des Studieng.:* ..... [1] Etudiant/Student:	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 13.05.2013 Remise du rapport / *Abgabe des Schlussberichts:* 27.09.2013 Défense orale / *Mündliche Verfechtung:* Semaine | Woche 40

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*

Durch seine Unterschrift verpflichtet sich der Student, die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Force Controlled Haptic Experiment With Two Different Robots

Graduate Summermatter Franco

Objectives

Realisation of a force controlled haptic experiment setup, which covers the system identification process for the used robots and the implementation of two different master slave operating scenarios with MATLAB/Simulink.

Methods | Experiences | Results

Two robots with different abilities are involved within this experiment, they build a haptic master-slave control system. Two operation scenarios were possible within this system were always one robot controls the other. Simulink acts as interface between the robots. Each scenario consists of a diagram for the simulation and a diagram for the real plant experiment.
To obtain the models used within the simulation diagrams, software tools were implemented which cover the different steps of the system identification procedure. Grey box modelling is used which permits to define precise linear or nonlinear model structures. The parameter for these models can be valued trough an estimation algorithm which is able to treat nonlinear plants.
The simulation and the real plant experiment can be actuated by the same artificial command signal, their output was compared and the models validated.
The main purpose of the developed Simulink framework is to provide a tool to design more advanced controller setups within the experiment environment.

First Robot: SensAble Phantom Omni Haptic Device. This root offers 3DOF force feedback and 6DOF position encoding.

Second Robot: Quanser 2DOF serial flexible joint robot. Each joint has one position encoder, one for the motor and one for the arm.

Bachelor's Thesis
| 2013 |

Degree course
Systems Engineering

Field of application
Infotronics

Supervising professor
Dr Chris J.B. Macnab
cmacnab@ucalgary.ca

Partner
Schulich School of Engineering,
University of Calgary
Alberta, Canada

TABLE OF CONTENTS

FORCE CONTROLLED HAPTIC EXPERIMENT WITH TWO DIFFERENT ROBOTS

1 INTRODUCTION

The presented work is about a haptic experiment setup which includes two different types of robots. The first robot is a SensAble Phantom Omni, the second is a Quanser 2DOF[1] robot with flexible joints (2DSFJ). Both robots are connected to the same PC and Simulink is used in external mode to control the robots. Two control scenarios are possible, as the master and the slave robot may switch their roles. The scenarios are defined as follow:

- Scenario 1: The Phantom Omni is the master, the 2DSFJ robot the slave.
- Scenario 2: The 2DSFJ robot is the master, the Phantom Omni the slave.

Each scenario has its own Simulink versions, one for the real plant experiment and one for simulation purposes.

The presented work covers beside the system description three major chapters: System identification, Scenario 1 and Scenario 2. The System identification chapter encloses all steps from data acquisition to model verification. Its goal is to find models, which represent the robots in the simulation diagrams. The implemented software tools are explained alongside an example. The Scenario 1 and 2 chapter covers the designed Simulink diagrams and their use for both scenarios. Simulation and real plant experiments use exactly the same controller, this makes is easy to simulate a controller design before implementing and testing it on the real plant.

Within this work, only a basic controller is implemented which fulfills the task. It is intended as performance reference for more advanced controller designs. These controllers can be simulated and tested within the presented Simulink framework.

1.1 2 Channel Haptic Operation Principle

A haptic system for teleported robots consists of a master and a slave robot. In both scenarios mentioned above, the operator applies a force on the master in order to achieve a proportional, desired force on the slave. The environmental force of the slave is feed back to the master. The operator feels the resistance of the slave against the movement on the master device dependant on obstacles or the environmental viscosity. This feedback force is proportional to the environmental force.

[1] DOF Degree of freedom

1.2 The Control Problem

The control problem involves the tracking of the master commands within two extreme situations: Moving in free space and maintaining contact with a hard surface. The transition from one condition to the other should happen smoothly and without excessive force. The control scheme follows the force control principles presented in [1], were the command tracking is based on an augmented error which includes force and velocity signals.

No explicit torque or force sensors are installed on this system. This means that only the position measurements and the different robot abilities are used to obtain the force control.

1.3 Tasks

The following tasks have been performed:

- Setting up the hardware.
- Implementation of Simulink diagrams which represent the hardware.
- Implementation of software tools to build models of the robots. These models are used to simulate the system.
- Implementation of Simulink diagrams for two operation scenarios. Each scenario has a simulation and a real plant representation.
  - Controller design for both scenarios.
  - Verification of the system performance.

2 SYSTEM DESCRIPTION

This chapter contains a summary over the hardware used within the system. The robots are described here more in detail and their Simulink representations are presented. For the detailed hardware installation procedures, please refer to the corresponding operation manuals [2] and [3].

2.1 Overview

The system comprises the following main components:

- SensAble PHANTOM Omni
- PC with installed Software:
  - MATLAB R2011a, Version 7.12.0.635
  - Simulink with Quanser QUARC Toolbox, Version 2.2.1
- Q8 HIL[2] PCI Board
- Q8 Terminal Board
- AMPAQ, Two Channel current amplifier
- Quanser 2DSFJ Robot

The following figure shows how these components are connected together. These connections don't depend on the operation scenario and are always the same.

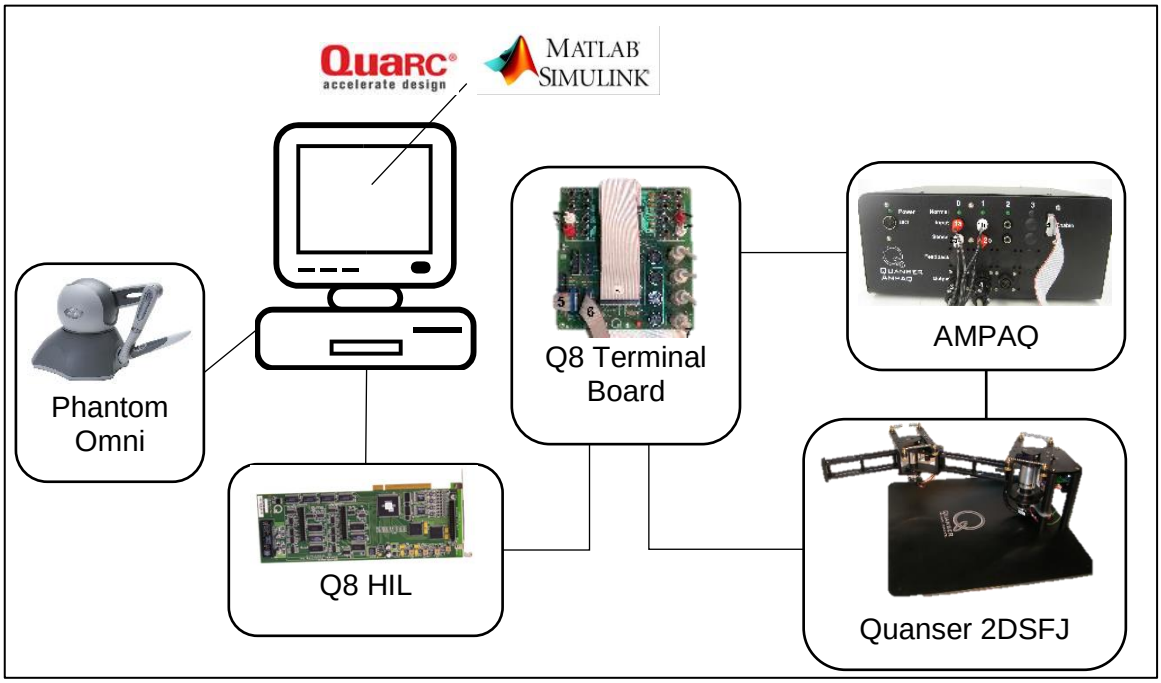


Figure 1: Deployment diagram of the system.

[2] HIL Hardware-In-the-Loop

2.2 Notations

To avoid confusion, the following definitions have been made in order to distinguish between the Phantom Omni and the 2DSFJ robot:

- Phantom Omni: the two controlled arms are named Y-Axis for the first and Z-Axis for the second arm. The angular position uses the Greek letter φ.
- 2DSFJ robot: the two controlled arms are named Stage1 and Stage2. The angular positions use the Greek letter θ.

The relation between master and salve robot arms is in every scenario:

- Stage1 ↔ Y-Axis.
- Stage2 ↔Z-Axis.

All MATLAB / Simulink files are named using the following prefixes for better structuration:

- A_... Refers to files, associated with the real plants (hardware operation).
- B_... Refers to files, associated with simulations (no hardware used).
- AB_... Refers to files, associated with the real plants OR simulations.
- C_... Refers to files, associated with system identification.
- D_... Refers to files, associated with verification.

To distinguish between the robots, the following midsection is used in filenames:

- …_2DSFJ_... Refers to the Quanser 2DSFJ robot.
- …_Omni_... Refers to the Phantom Omni haptic device.

And to distinguish between the scenarios:

- …_Scen1_... Refers to the Scenario 1.
- …_Scen2_... Refers to the Scenario 2.

2.3 SensAble PHANTOM Omni Haptic Device

This haptic device (Figure 2) has 6-DOF position encoding which means 3-DOF for the three main axis (Cartesian or angles) and 3-DOF for the gimbal (angles). No force measurement sensors are installed on this device. The following subsections described the integration of the hardware into usable Simulink blocks.

IMPORTANT NOTE: Before performing any powered operations, be sure the Phantom Omni has been calibrated![3]

Figure 2: SensAble PHANTOM Omni [2], the force feedback abilities are 3-DOF for the mentioned main axis only.

2.3.1 Axis and sign convention

The Phantom block (Figure 3) from the QUARC library builds the core of the Omni Simulink representation. This block can be parameterized for several types of haptic devices. It has the following sign definitions:

- X-axis: Left/Right movements, where Left is the positive direction.
- Y-axis: Up/Down movements, where Up is the positive direction.
- Z-axis: In/Out movements, where Out is the positive direction.

NOTE: The torque vector input has the same sign as the angles. That means i.e. a positive torque input on the y-axis will move the arm upwards.

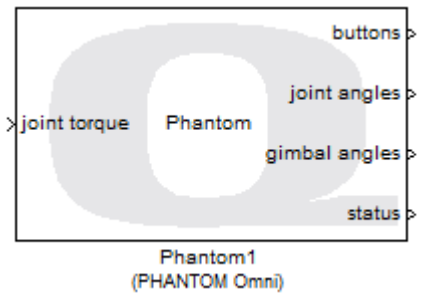


Figure 3: PHANTOM block, from the library *QUARC/Targets*

[3] To calibrate the Phantom Omni, run the program *PHANToM Test*

2.3.2 Horizontal operation mode

In order to have a more intuitive control experience and to avoid the gravity effect in the second scenario, the Omni has been put horizontally (Figure 4).

The sign definitions remain the same for the Y- and Z-Axis: CCW movements are read as positive.

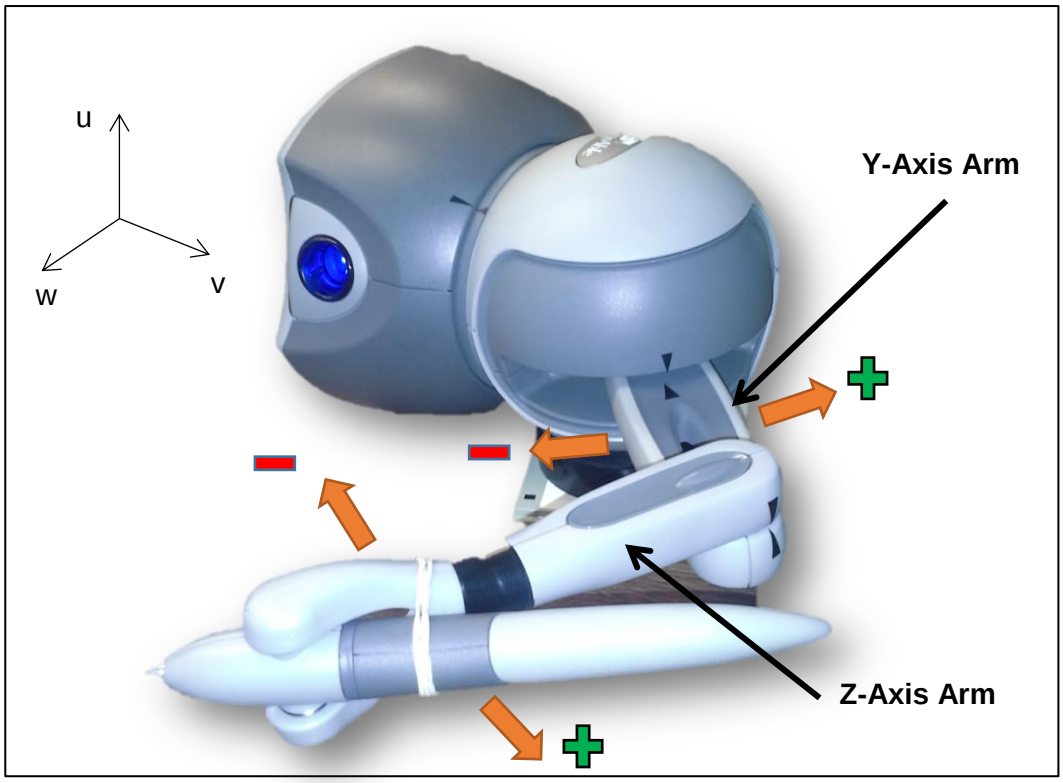


Figure 4: Phantom Omni put horizontally. The neutral positions are indicated by black marks on the device casing.

Due to the internal construction of the Omni, a compensator block is necessary to compensate for the following effects while the device is put horizontally:

- Y-Axis: internal mechanical spring pulls the arm in positive direction.
- Z-Axis: arm movement drifts in one direction.
- X-Axis: must be blocked to keep the arm in a level position.

Figure 5 shows the internal layout of this compensator block. A PI-Controller on the X-Axis keeps the arms level, the mechanical spring is compensated by a fictitious spring on the Y-Axis and a constant torque is applied on the Z-Axis to overcome its drift.

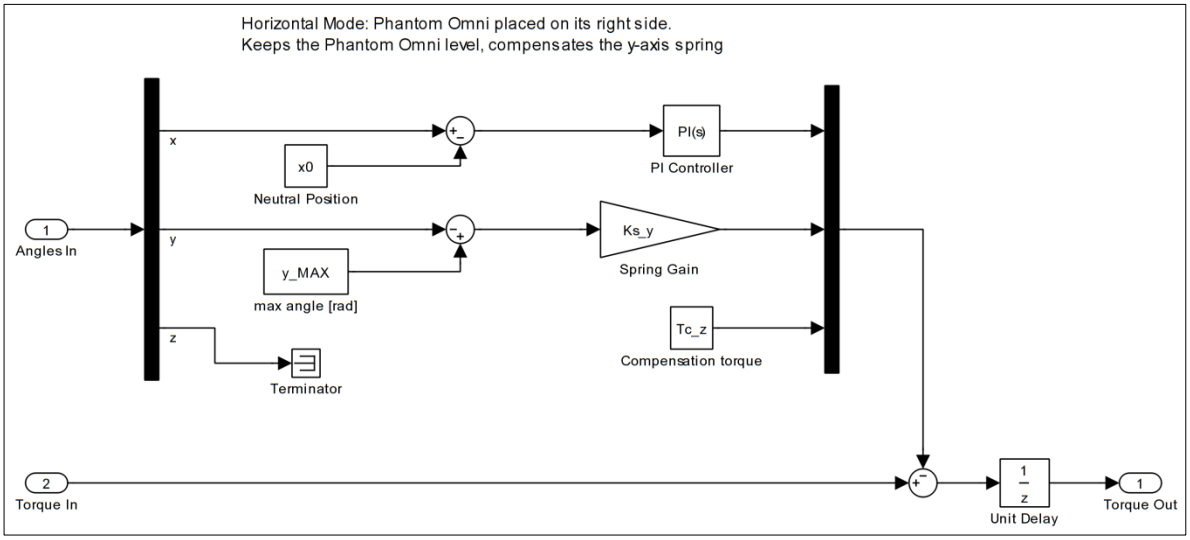


Figure 5: Compensator block layout, reduces the undesired effects when the Omni is put horizontally to a minimum.

2.3.3 Neutral position correction

Per default, the joint position angles which refer to the planes defined in Figure 4 have a range of:

- Y-Axis arm: 2 – 102 degree absolute referred to *uw*-plane.
- Z-Axis arm: 105 degree, referred to *uv*-plane. The limits of the Z-Axis are not absolute as they depend on the position of the Y-Axis arm.

NOTE: Due to the internal construction of the Omni, by moving the first arm around the Y-Axis, the measured position of the second arm is not affected. Actually, the device moves the Z-Axis as well in order to keep the position on the second arm.

The angle offset correction block (Figure 6) is used to convert the measured, pure positive range to a positive or negative range around a neutral position.

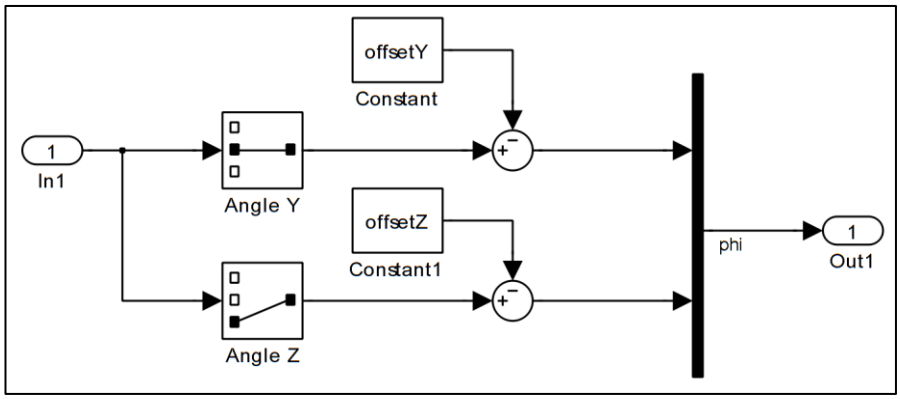


Figure 6: Angle offset correction block, shifts the zero on both axis to obtain a ± range.

2.3.4 Velocity filter

To get the velocity measurement, the position signal coming from the Omni is derivated and low pass filtered. This happens inside the velocity filter block for both axes (Figure 7).

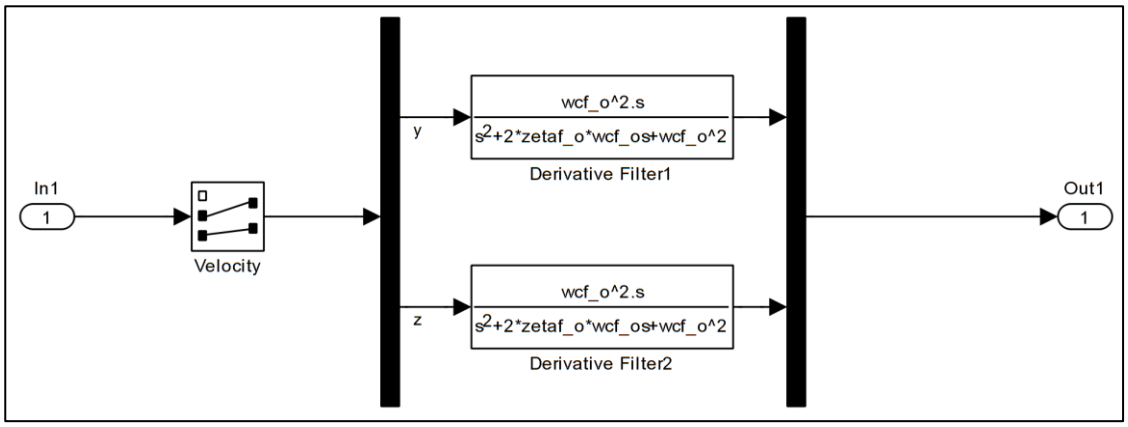


Figure 7: Velocity filter block, get the velocity out of position measurements.

2.3.5 Phantom Omni top level

All subsystem blocks mentioned in the previous subsections compose together the top level layout of the Phantom Omni block (Figure 8). This block is the base for all hardware depended Simulink diagrams. All parameters used are defined in the script `phantomOmni_setup` (code in Appendix 29). This script has to be executed to load these parameters into the workspace.

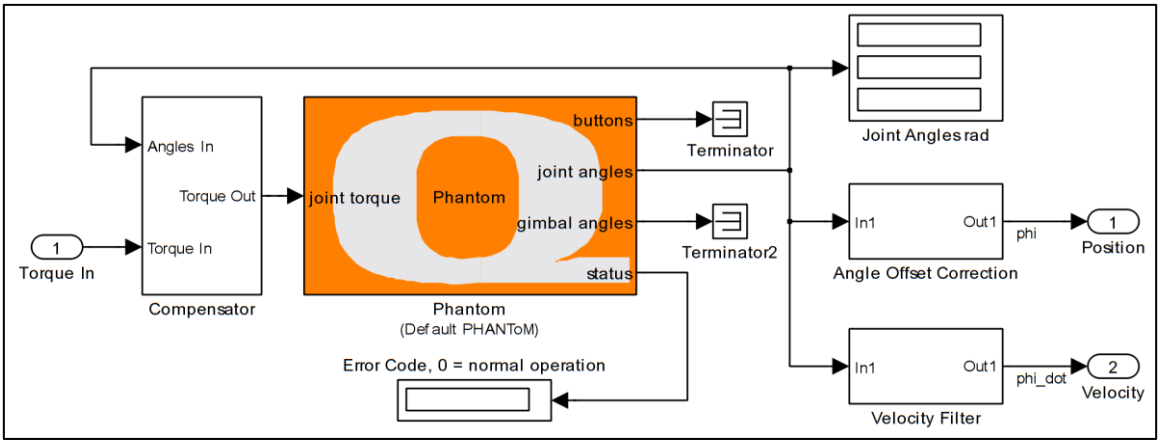


Figure 8: Phantom Omni block layout, takes a torque vector as input and gives a position and velocity measurement vector as output.

2.4 Quanser 2 DOF Serial Flexible Joint Robot (2DSFJ)

This robot (Figure 9) consists of two identical flexible joints. Each joint is equipped with a drive motor, a drive encoder and a joint encoder. The mechanical coupling between the driven part and the arm is realised with two compression springs. Each stage has its own harmonic drive and motor specifications. In the actual setup, the lightest springs are installed on both stages.

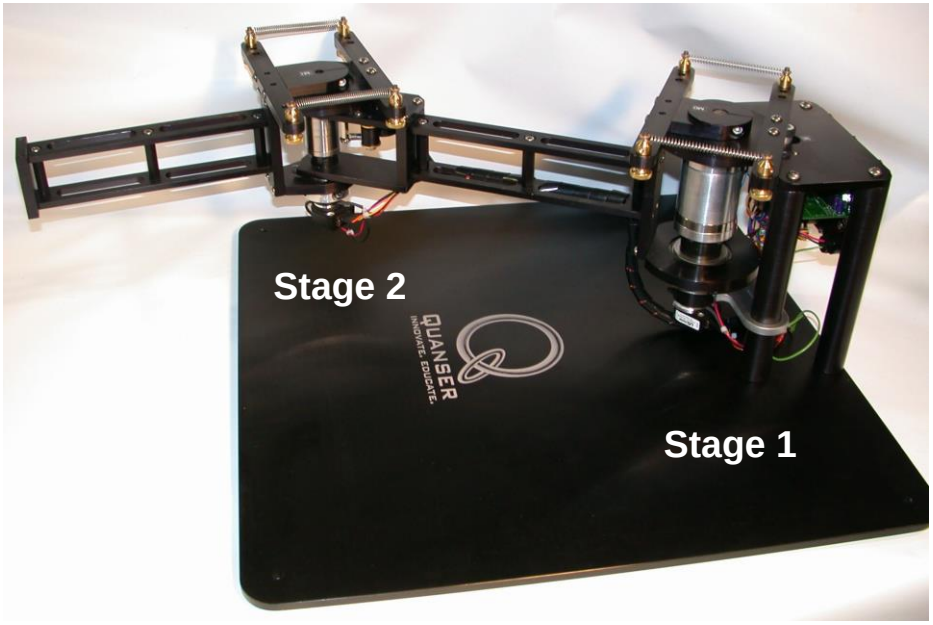


Figure 9: Quanser 2DSFJ Robot [3]

2.4.1 Stages and sign convention:

The stages are defined in Figure 9, the shoulder joint is named Stage1 and the elbow joint Stage2.

- For both stages, CCW is the positive direction viewed from the Top.

NOTE: When the robot moves in positive direction (CCW) and hits an obstacle, then the applied force to this obstacle is positive. This means a CW deflection of the flexible joints is read as positive force.

NOTE: otherwise than stated in [3], the actual gear ratio of the harmonic drive #2 is 80:1 instead of 50:1.

2.4.2 Verification of the position sensors

The gear ratio of the flexible joint encoder Kg_{enc} was set to 6.23 in order to achieve an angular accuracy of less than 0.2º between the drive and the joint encoder on both joints. This accuracy was measured on ±80º absolute angle which means a max. relative angle error of ±0.25% within ±90% of the joint's range.

2.4.3 Verification of the joint torsional stiffness K_s

The joint torsional stiffness K_s was verified to check which springs are installed and to have a physically coherent reference for the system identification process. This could be done due the following relationship between the torque applied from the environment T_{env} and the angle deflection $\Delta\theta$:

$$T_{env\,i} = K_{s\,i}\ \Delta\theta_i \qquad (1)$$

Where K_s and $\Delta\theta$ are defined as:

$$K_{s\,i} = 2\,K_{r\,i}\,r_i\,d_i \qquad (2)$$

$$\Delta\theta_i = \theta_{i2} - \theta_{i1} \qquad (3)$$

The index i references the corresponding joint. K_r stands for the installed spring rate, r is the radius from the joint rotational axis to the hole where the spring is installed and d is the distance from the joint centerline to the hole. The angle deflection (3) is the difference between the joint encoder 2 and the drive encoder 1 which complies with the signs defined in 2.4.1.

In order to verify the theoretical value for K_s found with (1), a known torque T_{env} was applied to each joint and the angle deflection was measured with Simulink. An average value for K_s could be calculated. The method used was to hang on different weights at a given distance to the flexible joint rotational axis.

The theoretical torsional joint stiffness for both joint with the lightest spring installed in the other hole is 3.926 [Nm/rad]. The measured average was 3.971 [Nm/rad] for the joint #1 and 3.891 [Nm/rad] for joint #2. The measurements are listed in Appendix 1.

K_{s1} respective K_{s2} are now verified and could be seen as fix parameters for the system identification process.

2.4.4 Simulink interpretation

The robot is connected to Simulink over the PCI board Quanser Q8 HIL. The QUARC toolbox contains specific blocks to communicate with the Q8 board. Basically, this communication contains: reading the position encoders, reading the actual current and writing the commanded current to the current amplifier.

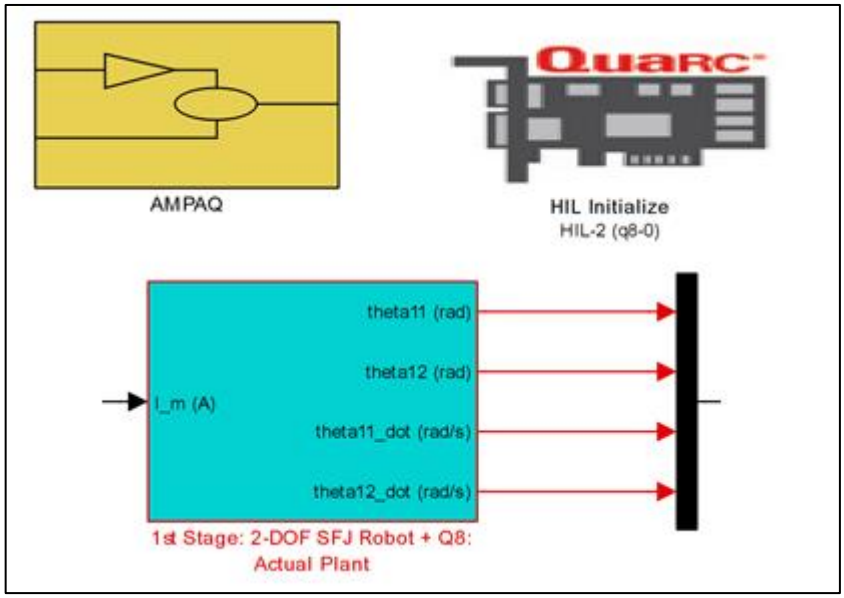


Figure 10: Simulink blocks: AMPAQ, HIL initialize and one stage of the 2DSFJ robot.

The general layout of one robot stage (Figure 10, lower block) is shown in Appendix 13. The basic layout was provided by an application example from Quanser it was adapted to fulfil the task. Additionally to the reading/writing operation, it consists of:

- a current limitation,
- a current correction to overcome the drift in one direction,
- a sign converter for the positive rotation direction,
- two second order low pass filters to obtain the angular velocities.

NOTE: In order to use the Simulink block described above, the block QUARC HIL initialize and AMPAQ must be present as well (Figure 10). These blocks contain the initialisation settings for the communication and the position limitation safety feature.

NOTE: The maximal continuous current after [3] is 0.94A for Stage1 and 1.2A for Stage2. However, these are max. values and it is recommended not to exceed 0.3 - 0.5A. As the robot movement will already be very quick.

All parameters necessary by the real plant related Simulink blocks are defined in the script `robot_2DSFJ_setup` (code in Appendix 28). Run this script to load these parameters into the workspace.

2.4.5 Safety feature

The 2DSFJ robot consists of two HAL sensors per stage. These sensors are located at the maximum and minimum physical position of the robot arm. If this safety is enabled (`LIMIT_SWITCHES_ENABLE = 1`) and one of the sensors is triggered, the execution of the running experiment will be stopped and an error message will be displayed.

The monitoring over these switches is included in the AMPAQ block provided from Quanser, see Appendix 14 for more details.

2.5 Signal Generator GUI

The signal generator GUI (Figure 11) is a key tool used for the experiment and simulation execution. It permits to simulate the master commands, dependant on the context this may be a position or a current.

All actuation signal parameter can be altered inside the GUI and a visual preview of these signals is offered. The user knows exactly which stage is enabled and how it will be actuated before the experiment/simulation execution. Additionally, the execution duration of the currently linked Simulink diagram can be altered directly in the GUI. The field *Signal Delay* offers the option to delay the actuation signal. This is useful when actuating real plant experiments in order to power up the robots first and start the actuation some time later. The simulation duration in the Simulink diagrams is set to a value: *Simulation Duration + Signal Delay*.

NOTE: The duration of the preview is the same as the simulation duration, the signal delay is not taken into account. If the simulation duration is set to 'inf' a maximum timespan of 100s is displayed.

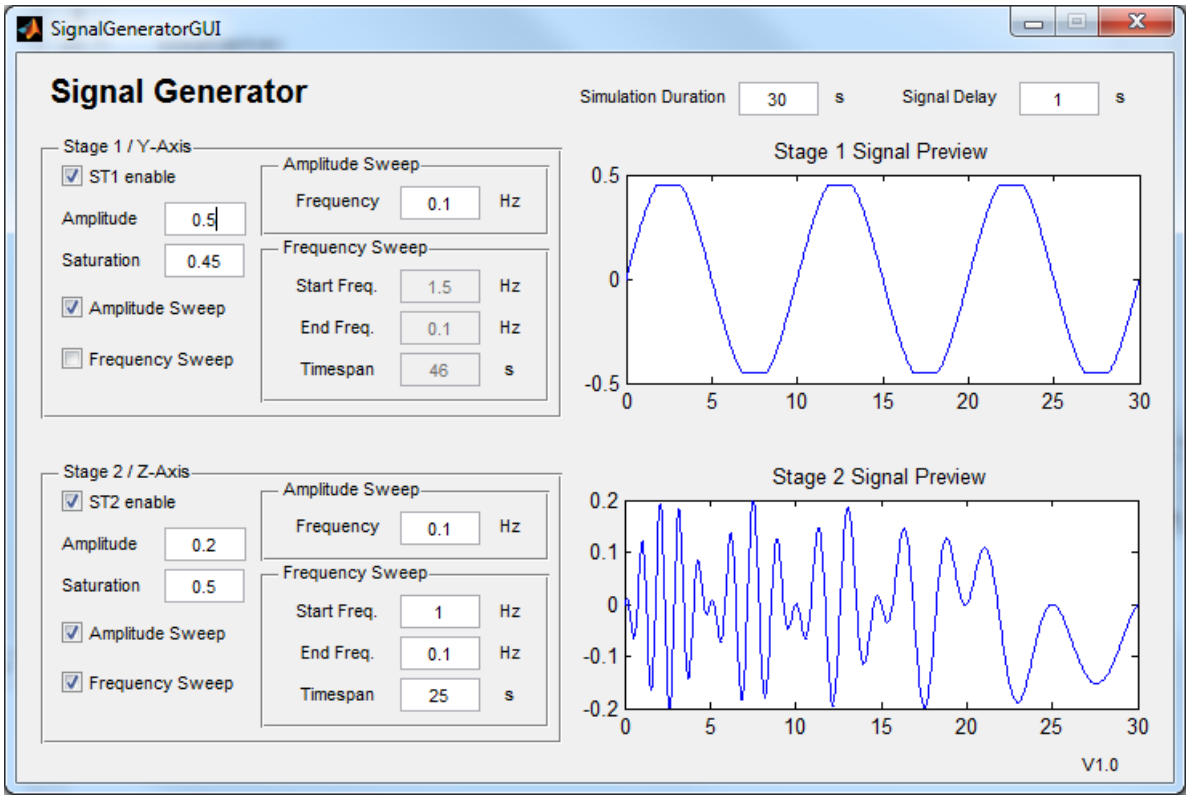


Figure 11: The signal generator GUI provides a preview of the actuation signal before running the experiment. This saves time and inhibits surprised actuation signal patterns within the experiment.

The following actuation signals or their combination can be generated, the signals may be subject to user defined saturation:

- Step, with selectable amplitude.
- Amplitude sweep. (e.g. Figure 11, Stage 1 preview with saturation)
- Frequency sweep (chirp function). (e.g. Figure 11, Stage 2 preview with amplitude sweep)

The signal generator GUI is always connected to a main script which comprises the default parameters and the name of the linked Simulink diagram. By executing the main script, the default parameters are loaded into the MATLAB workspace, the linked Simulink diagram and the signal generator GUI open.

Altering parameters within the signal generator GUI changes only their values inside the workspace. The Simulink diagram takes the current values from the workspace while executing.

NOTE: By running the main script again, all altered parameters are overwritten with their default values assigned in the main script.

The following figure shows the Simulink representation of the signal generator block:

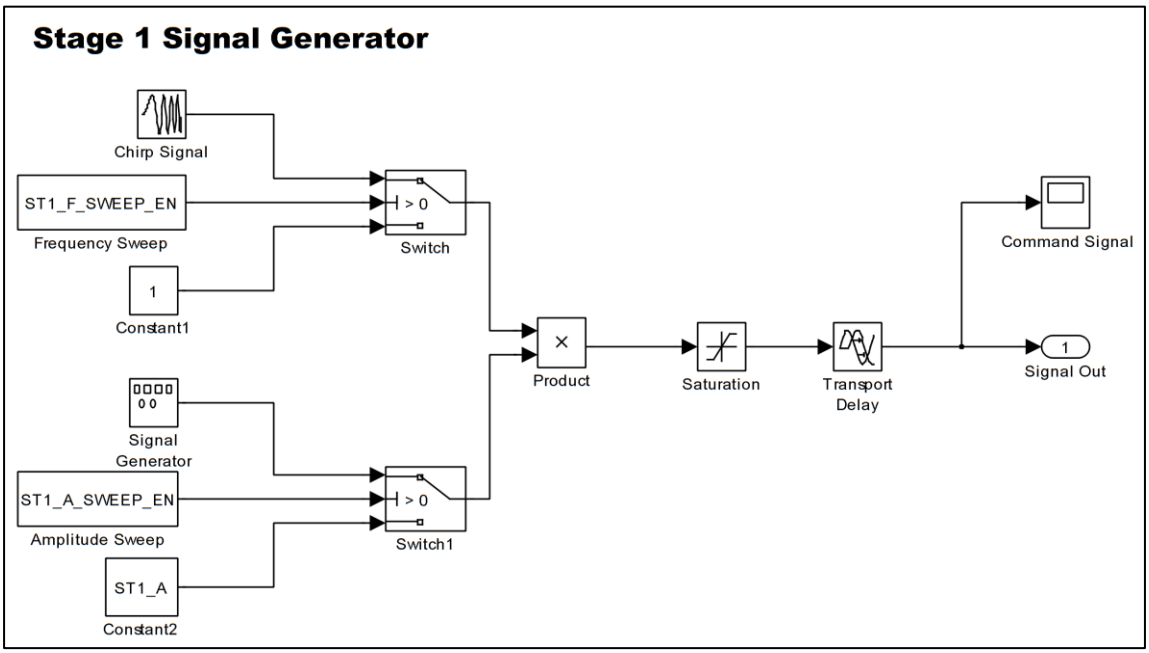


Figure 12: Signal Generator block. The signal generator offers a wide variety of actuation signal patterns, from a simple step to amplitude modulated frequency sweeps.

The MATLAB code for the signal generator GUI was mainly auto-generated by MATLAB's GUIDE[4] tool. Therefore only the code for the opening function and the two main functions is attached in Appendix 30. The GUI concept is based on two functions, dependant on the field altered, one of these functions is called:

- **updateFields:** This function allows access to the different fields by enabling or disabling them if the corresponding checkbox is ticked. E.g. the amplitude sweep frequency setting may only be accessed if the checkbox *Amplitude Sweep* is ticked.
- **updatePreview:** This Function reads in all current values of the GUI and updates the values of the corresponding parameters in the workspace. It shows the calculated preview of the actuation signal. Every field related to signal settings calls this function if changed.

[4] **G**raphical **U**ser **I**nterface **D**esign **E**nvironment

3 SYSTEM IDENTIFICATION

The purpose of the system identification is to find a model which can be used to simulate the systems behaviour for the controller design without the need of any hardware. As more accurate the model matches the real system, as more complex it will become. The model is always a trade-off between complexity and accuracy. Therefore simplifications are welcome but only if they affect the accuracy in an acceptable manner. This chapter explains the use of the implemented software tools to build models from data collection to model validation for each plant alongside an example.

The system identification process consists of the following steps:

- Experiment data acquisition.
- Model Structure definition.
- Parameter estimation.
- Model validation.

The system contains two independent plants, the Quanser 2DSFJ robot and the SensAble Phantom Omni haptic device. Each of them has its own characteristic and is treated independently with their own diagrams and parameter setups to obtain the models. However, the actuation signals are in both cases provided by the signal generator block.

There are several different procedures possible within the MATLAB System Identification Toolbox. Even though the model structure may be linear, the nonlinear grey box modelling was chosen because of the following advantages:

- Estimation according to parameters, (not location in state-space matrices).
- Individual parameters can be set to be fixed without touching the model structure.
- Flexible to adapt to additional parameters or another structure.
- Ready to treat nonlinearities.

However, the initial set up needs more effort because the model structure must be implemented in a function.

3.1 Quanser 2DSFJ Robot

This section covers the system identification steps for the 2DSFJ robot. The two stages are treated individually. However, all scripts and diagrams are common for both stages. One stage can always be disabled while the other is treated.

As the steps don't differ from one stage to the other, only the procedure for Stage1 is mentioned here more in detail. The following table shows all MATLAB files associated with this section:

File Name	Description	Location Folder
`constants.m`	Contains general constants definitions, Appendix 27.	0_Parameters
`robot_2DSFJ_setup.m`	Basic parameters needed for the real plant, Appendix 28.	0_Parameters
`SignalGeneratorGUI.m`	Function for the signal generator operation, Appendix 30.	0_Parameters
`C_2DSFJ_Data_Acq_main.m`	Main script for the data acquisition experiment, Appendix 31.	3_2DSFJ_Model_Est
`C_2DSFJ_Data_Aqu.m`	Script to build the *iddata* objects out of the raw data, Appendix 32.	3_2DSFJ_Model_Est
`C_2DSFJ_Model_Est_main.m`	Script for the parameter estimation process, Appendix 33.	3_2DSFJ_Model_Est
`C_2DSFJ_Data_Aqu_SM.mdl`	Simulink diagram, represents the experiment in open loop mode, Appendix 15.	3_2DSFJ_Model_Est
`C_2DSFJ_Model.c`	Function containing the model structure definition, Appendix 34.	3_2DSFJ_Model_Est
`C_2DSFJ_DataObjects.mat`	Contains the saved *iddata* objects.	3_2DSFJ_Model_Est
`C_2DSFJ_Data_Aqu_SM.mat`	Contains the raw data of the last experiment. Overridden each time.	3_2DSFJ_Model_Est
`pars_2DSFJ_stage1.mat`	Contains the initial guessed parameters for Stage1.	3_2DSFJ_Model_Est
`pars_2DSFJ_stage2.mat`	Contains the initial guessed parameters for Stage2.	3_2DSFJ_Model_Est
`C_2DSFJ_ModelObjects.mat`	Contains the saved estimated *idnlgrey* model objects.	3_2DSFJ_Model_Est

Table 1: MATLAB files used within the 2DSFJ system identification process.

3.1.1 Experiment Data Acquisition

1. Selection of the actuation signal

The quality of a model is highly dependent on the data used for the estimation process. Therefore a good experiment data set or sets are needed to perform successful parameter estimation. The selection of the actuation signal which guarantees as much useful information as possible but without exceeding the plants physical limits is crucial. Several tests revealed, that high amplitude high frequency signals are best for the first estimation. This is especially important for plants with high dry friction.

To start a data acquisition experiment, open the MATLAB script `C_2DSFJ_Data_Acq_main`. On top of the script, the default parameter for the experiment can be altered.

NOTE: The parameter `SIM_DELAY` acts as transport delay for the actuation signal. The purpose of this delay is to avoid peak currents at the start of the experiment which may occur when the robot is powered and the actuation signal starts at the same time. Another reason is to start the experiment with initial states close to zero. This delay is removed automatically form the captured data in the next step.

By running the script, the Simulink diagram `C_2DSFJ_Data_Aqu_SM` and the signal generator GUI open. The Simulink diagram operates the 2DSFJ robot in open loop mode, the input signals amplitude unit is Ampere. **Special care should be taken with amplitudes above 0.1A for the first stage and 0.2A for the second stage, as the response may be very quick!**

NOTE: By changing the simulation duration, a rebuild of the Simulink diagram is necessary. This can be done by selecting the open diagram and pressing *Ctrl + B* or selecting within Simulink the menu bar QUARC > Build.

2. Run the Experiment

With the actuation signal set, the experiment can be executed as usual within Simulink. The experiment data for both stages is logged into the file `C_2DSFJ_Data_Aqu_SM.mat`.

Two demo data sets were collected for each stage. The actuation signal settings for these data sets can be found in Appendix 3.

NOTE: The experiment output file is saved in the current open folder in the file browser. In order to avoid error messages in the next step, the experiment should be run with the folder *3_2DSFJ_Model_Est* open, where the execution file is located.

NOTE: Do not alter the values in the signal generator GUI between the experiment execution and the data set build in the next step. The script `C_2DSFJ_Data_Aqu` uses the current values from the workspace. By changing these values in the meantime, the information stored in the *iddata* objects will be false.

3. Analyse the Experiment Data

After the experiment, the script `C_2DSFJ_Data_Aqu` provides treatment and analysis of the captured raw data. On top of the script, the dataset name which will be stored within the *iddata* object cell can be defined.

By running this script, the *iddata* objects are created according to the experiment settings, this includes the following steps:

- Remove the simulation delay.
- Extract the raw data input and output values for the enabled stages
- Build the data sets: Assign names, units, etc.
- Add information about the experiment according to the actuation signal settings. This information is displayed on the MATLAB console.

Next, the System Identification GUI opens (Figure 13). With this tool, the build *iddata* objects can be imported and analysed easily.

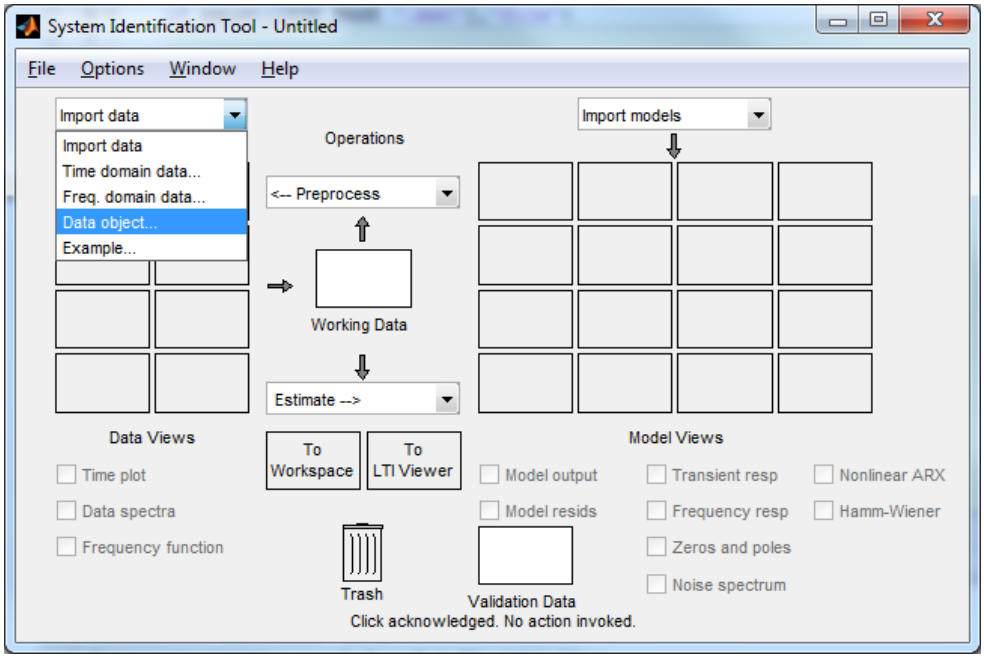


Figure 13: System Identification GUI, to import a new data set, just select on the **Import data** drop down menu the entry **Data object**.

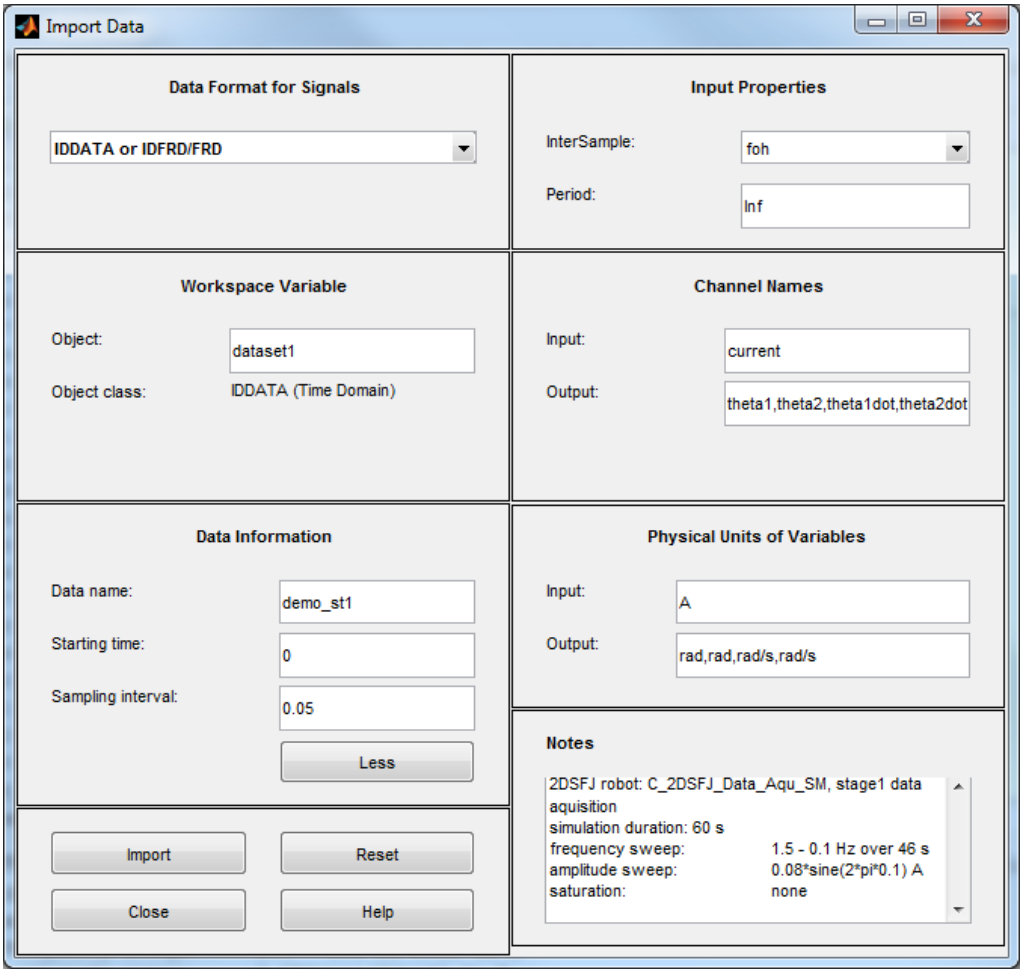


Figure 14: Import Data Pop up Window, clicking on **More** shows information about the data set.

Enter on the **Import Data** pop up window (Figure 14) the workspace variable name of a data set to be imported and by clicking on **Import**, the dataset appears on the left hand side of the System Identification GUI.

To analyze the imported data, check the box **Time plot** which opens a new window (Figure 15), displaying each input / output channel in the time domain.

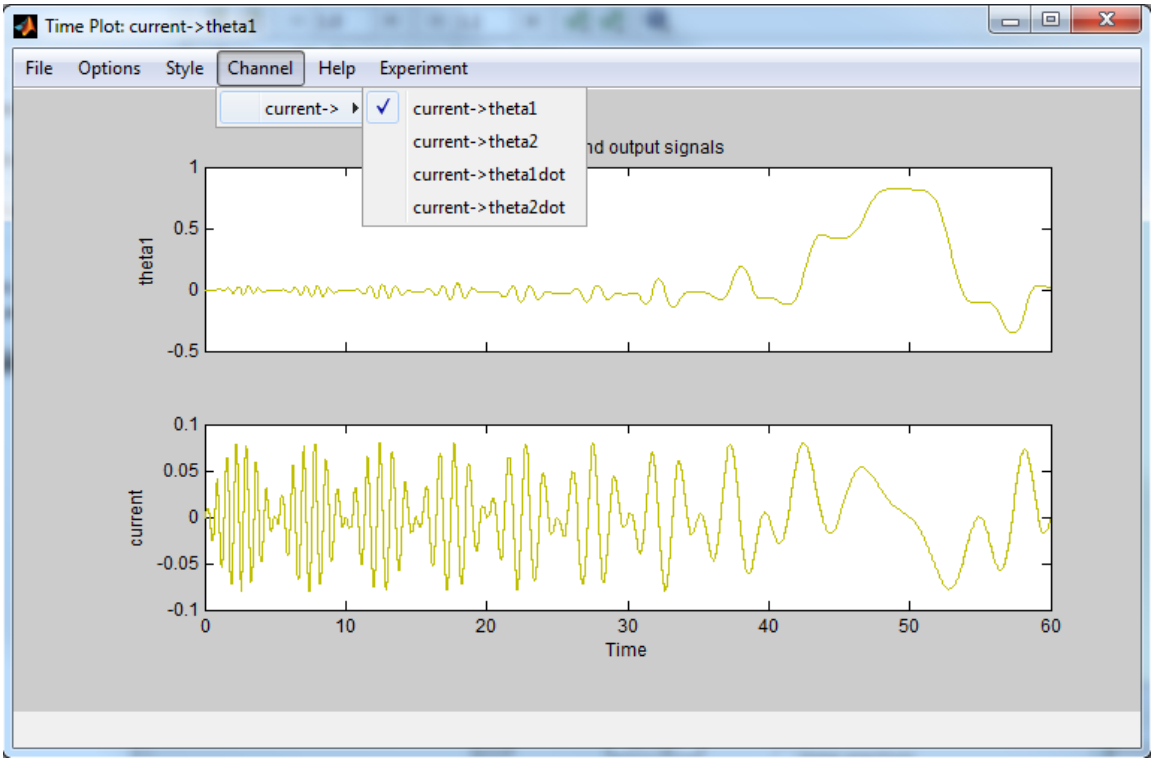


Figure 15: Time plot, select under **Channel** every input/output combination. **Experiment** allows switching between the stages.

4. Store the Data Object

If the captured data seams correct and useful, the datasets must be stored manually in order to proceed with the model estimation process. The file `data_2DSFJ.mat` is meant to contain the saved *iddata* objects. To save a data set, rename it in the workspace and simply drag and drop it into the file.

NOTE: Data sets can also be exported to the workspace form the System Identification Tool GUI. This is useful if additional treatments like de-trending, resampling or another range etc. was applied within the GUI.

3.1.2 Definition of the Model Structure

With the experiment data ready for parameter estimation, it is now the time to define the model structure. As simpler the model structure as faster the parameter estimation process but on the cost of accuracy. On the other hand, if the structure becomes too complex with many parameters, it is more difficult to achieve good results. This is especially the case if many or all initial parameters are unknown.

The chosen linear model structure for the 2DSFJ robot results from its physical description. The complete, simplified physical description is shown in Figure 16. The simplification consists of the hypothesis, that the drive #2 is installed on the center of gravity of the first stage. As each stage has its own independent position measurement, the system can be seen as decoupled and this hypothesis is reasonable. Please refer to Appendix 2 for the coefficient explanation and their corresponding initial values were applicable.

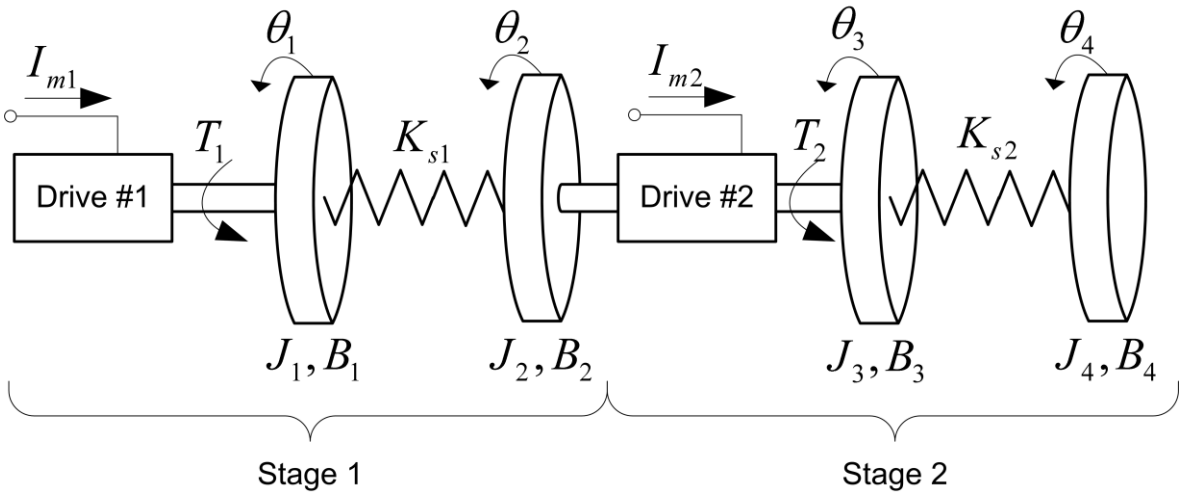


Figure 16: Complete physical schematic of the 2DSFJ robot [3].

Single Stage Model

After decoupling, the system can be reduced to two stages with identical differential equations. Therefore, only the first stage is mentioned here more in detail. However, the parameter values of each stage are different to comply with the real system.

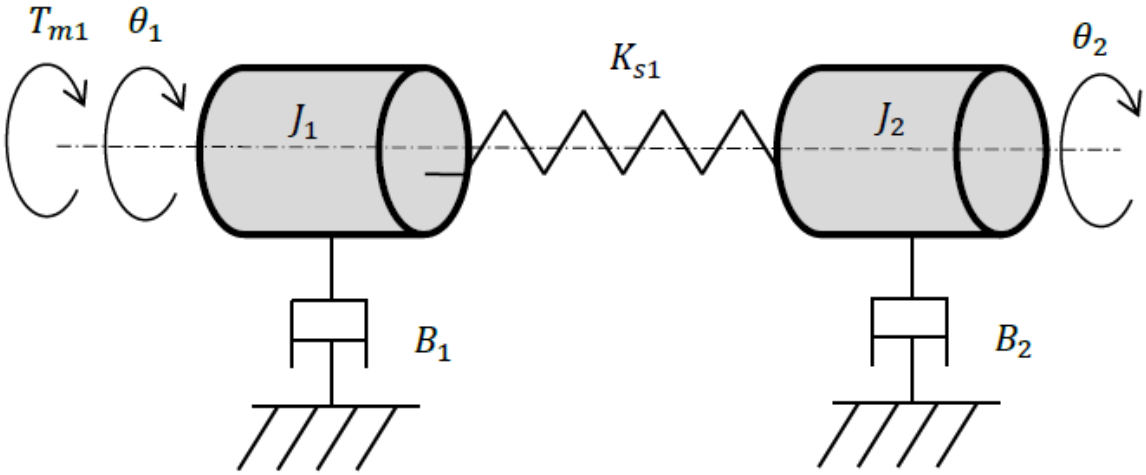


Figure 17: Physical schematic of Stage1.

The schematic above gives the following system of motional differential equations for the two moving bodies:

$$\begin{matrix} J_{11}\ddot{\theta}_{11} + B_{11}\dot{\theta}_{11} + K_{s1}\theta_{11} & -K_{s1}\theta_{12} & = T_{m1} \\ -K_{s1}\theta_{11} & +J_{12}\ddot{\theta}_{12} + B_{12}\dot{\theta}_{12} + K_{s1}\theta_{12} & = 0 \end{matrix} \quad (4)$$

With the applied motor torque defined as:

$$T_{m1} = K_{tg1}\, I_{m1} \quad (5)$$

State-Space representation

These two differential equations are second order with two independent variables, the two angular positions θ_{11} and θ_{12}. This SIMO[5] model can be represented in state space form using the general state-space equations (6).

$$\begin{aligned}\dot{\vec{x}} &= A\vec{x} + B\vec{u} \\ \vec{y} &= C\vec{x} + D\vec{u}\end{aligned} \qquad (6)$$

The two second order differential equations (4) and can be transformed into 4 first order differential equations by defining the state variables as the two angular positions and angular velocities of the two bodies. The state, input and output vector is defined as follow:

$$\overrightarrow{x_1}^T = [\theta_{11}\ \theta_{12}\ \dot{\theta}_{11}\ \dot{\theta}_{12}\ ] \qquad (7)$$

$$\overrightarrow{u_1} = [I_{m1}] \qquad (8)$$

$$\overrightarrow{y_1} = \overrightarrow{x_1} \qquad (9)$$

On the actual plant of the 2DSFJ robot, the two positions of each joint are measured with incremental sensors; the corresponding velocity is derivated from the position. Therefore all four state variables can be measured on the actual plant. By defining the output vector equal to the state vector, the measured and simulated values can be compared easily.

The system, input, output and feedforward matrices of the resulting state-space model for the first stage are:

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_{s1}}{J_{11}} & \frac{K_{s1}}{J_{11}} & -\frac{B_{11}}{J_{11}} & 0 \\ \frac{K_{s1}}{J_{12}} & -\frac{K_{s1}}{J_{12}} & 0 & -\frac{B_{12}}{J_{12}} \end{bmatrix} \qquad (10)$$

$${B_1}^T = \begin{bmatrix} 0 & 0 & \frac{K_{tg1}}{J_{11}} & 0 \end{bmatrix} \qquad (11)$$

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (12)$$

$${D_1}^T = [0 \quad 0 \quad 0 \quad 0\ ] \qquad (13)$$

The state-space matrices for the second stage are analog to them of the first stage.

[5] SIMO Single Input Multiple Output

Nonlinear Grey Box Model Structure

Grey box modelling in MATLAB necessities the implementation of a function containing the model structure related to the state-space representation. This means for a given time and given states as input, the calculation of the derivative states and the system output as result. These equations can be extracted from the state space representation above and are implemented as a state equation function and an output equation function in the file `C_2DSFJ_model.c` (Function code in Appendix 34).

The model structure function can be implemented with the MATLAB programing language or i.e. in C. The implementation in C was chosen here because of the faster program execution as there are many function calls within the estimation process.

But these C functions need a Gateway function (same as *main* in normal C, see Appendix 34 line 55ff) which acts as interface between MATLAB and C type definitions. This Gateway function was taken form a MATLAB tutorial and is not further explained in this document. The reader may find more information in the MATLAB documentation.

NOTE: MATLAB cannot use the C functions directly, these have to be compiled into C-MEX files. The MATLAB command *mex 'filename.c'* must be used for this purpose.

3.1.3 Parameter estimation

The model structure is defined, the grey-box model function implemented and compiled. Now, the estimation for the defined parameters can take place. According to the 2DSFJ model structure, there are six parameters to be estimated for each stage: Ktg, Ks, J1, B1, J2, and B2.

The prediction error estimation method pem is used to estimate the free parameters. There are several options available within this method (refer to the MATLAB documentation). One of the most significant is the setting of the initial states which can be `Zero` or `Estimate`. Dependant on the data set used and even if the initial states were zero, the selection of `Estimate` can give much better results.

Open the script `C_2DSFJ_Model_Est_main`. Several settings can be made for the estimation process:

1. Basic settings:
   - Select the stage to be estimated: i.e. `EST_AXIS = 1` for stage 1.
   - Assign the dataset for the estimation and the verification process, here:
     - `estData = demo_st1`
     - `verDat = demo2_st1`
2. Advanced settings:
   - Select the max. No. of iterations per data points (`MAX_ITER`). As higher the number as longer the process will take.
   - Select the minimal and maximal value for each parameter (`PARS_MIN`, `PARS_MAX`). The estimation function will respect these constraints.
   - Select the increment step in data points per iteration (`ITER_POINTS`) for the estimation loop. This value can be chosen e.g. dependent on the amplitude sweep period time.

Example: Amplitude sweep frequency 0.1Hz means zero crossing every 5s, with a sampling time of 0.05s, this results in 100 data points between zero crossings.

For one single estimation step for the entire data set, choose:

```
ITER_POINTS = NO_DATA_POINTS.
```

- Specify model structure function name, No. of parameters in the model structure and model order (`FUNC_NAME, PARS_NBR, ORDER`).

Every parameter in the model structure can be set to be estimated or not. To specify a parameter not to be estimated set the field `Fixed` to `true`.

Example: `grey2DSFJ.Parameters(2).Fixed = true;` Means, the second parameter will not be estimated and remain on its initial value.

By running the script, the estimation process is displayed in the console and the actual estimation result is shown in a new figure window dependant on the settings for `ITER_POINTS`.

3.1.4 Model Verification

If the defined model structure and the corresponding estimated parameters represent well the real plant, can be verified using the method `compare` from the System Identification Toolbox. This method takes an *iddata* object as reference and compares the output of one ore more models with this reference data. It is important when ever possible to use different data sets for the estimation and the verification process to guarante the model corresponds to the dynamic of the plant and not to the trends of a single data set.

Figure 18 shows the plot from such a comparison for stage 1, three models were compared:

- modSt1Init, using initial parameters without estimation.
- modSt1DampOnly, only the two damping coeffitients B1 and B2 were estimated, all others remained on their initial value.
- modSt1All, all parameters were estimated.

For all models (were applicable) the *iddata* object `st1DemoData` was used for the estimation process.

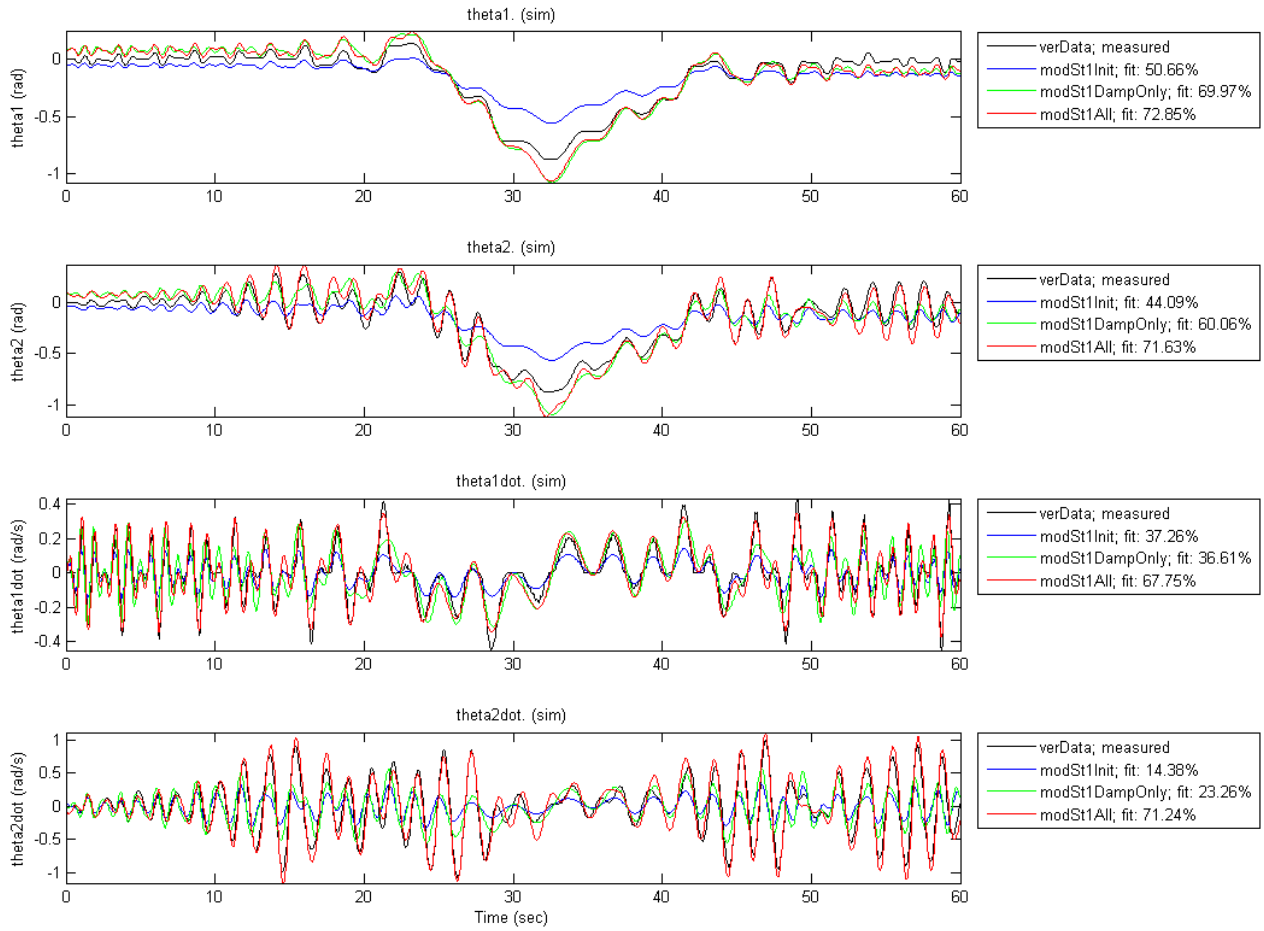


Figure 18: Stage 1 model verification demo, verData object was `st1DemoData2`

In every of the 4 measured outputs (Figure 18), `modSt1All` has the best match. It can be observed that `modSt1Init`, were all parameters remained on their initial values does not reproduce the real plants behaviour very well. Even though, this model uses parameter values very close to the physical properties (provided by Quanser). The estimated parameters of all models are listed in Appendix 4.

NOTE: As a linear model structure is used to match a real plant which always has certain nonlinearities, the estimated parameters are not to be meant as absolute physical properties. The estimated values must be interpreted as best combination to simulate the dynamic behavior of the plant. However, if a more accurate model in terms of physical properties is desired, more parameters must be fixed. But this will certainly affect the model performance.

The model `modSt2All` shows a good representation of the actual plant, the linear model structure is therefore acceptable.

One demo model (*st1DemoModel* and *st2DemoModel)* for each stage was estimated and saved in the file `C_2DSFJ_ModelObjects.mat`. Their parameter values and verification plots can be found in Appendix 5 and Appendix 6.

The models were estimated with the iddata objects *st1DemoData* respective *st2DemoData*. They were verified with the iddata objects *st1DemoData2* and *st2DemoData2*. The fit compared to the velocity signals of the verification data is:

- Stage1: 65.21% for the motor position and 72.71% for the robot arm position.
- Stage2: 62.66% for the motor position and 53.46% for the robot arm position.

3.2 SensAble Phantom Omni

In this section, the system identification steps for the SensAble Phantom Omni haptic device are described. The basic procedure is the same as in section 0, therefore only the differences are mentioned here. The following table shows all MATLAB files associated with this section:

File Name	Description	Location Folder
constants.m	Contains general constants definitions, Appendix 27.	0_Parameters
phantomOmni_setup.m	Basic parameters needed for the real plant, Appendix 29.	0_Parameters
SignalGeneratorGUI.m	Function for the signal generator operation, Appendix 30.	0_Parameters
C_ Omni_Data_Acq_main.m	Main script for data acquisition experiment.	4_Omni_Model_Est
C_ Omni_Data_Aqu.m	Script to build the *iddata* objects out of the raw data.	4_Omni_Model_Est
C_ Omni_Model_Est_main.m	Script for the parameter estimation process, Appendix 35.	4_Omni_Model_Est
C_ Omni_Data_Aqu_SM.mdl	Simulink diagram, represents the experiment in closed loop mode, Appendix 16.	4_Omni_Model_Est
C_Omni_model_2DOF_c.c	Function containing the linear model structure, Appendix 36	4_Omni_Model_Est
C_Omni_nl_model_2DOF_c.c	Function containing the nonlinear model structure, Appendix 37.	4_Omni_Model_Est
C_ Omni_DataObjects.mat	Contains the *iddata* objects	4_Omni_Model_Est
C_ Omni_Data_Aqu_SM.mat	Contains the raw data of the last experiment. Overridden each time.	4_Omni_Model_Est
pars_ Omni_ yAxis.mat	Contains the initial guessed parameters for the Y-Axis.	4_Omni_Model_Est
pars_ Omni_ zAxis.mat	Contains the initial guessed parameter for the Z-Axis.	4_Omni_Model_Est
C_2DSFJ_ModelObjects.mat	Contains the estimated *idnlgrey* model objects	4_Omni_Model_Est

Table 2: MATLAB files used within the Phantom Omni system identification process.

3.2.1 Experiment Data Acquisition

Other than the 2DSFJ robot, the Omni experiment is in closed loop configuration. This means the actuation signal form the signal generator will be a reference position. A P-Controller with gain 1 is used to control the position. This configuration is necessary due to the high influence of the dry friction.

As the data acquisition scripts are basically the same as used in 3.1.1, their MATLAB code is not attached to this document.

To start a data acquisition experiment, open the MATLAB script `C_Omni_Data_Acq_main`. The procedure and settings are the same as in 3.1.1, except of the following:

- Closed loop configuration: Actuation signal from the signal generator is a position reference in radian.
- P-Controller is present, gain values can be adjusted.
- Data acquisition script `C_Omni_Data_Aqu` contains an additional offset correction feature. The position measurement of the Omni is absolute. This means that the position at power up is not set to 0 as it was the case with the 2DSFJ robot. The position tracking starts at the position where the arms are hold manually. This is usually not exactly at the neutral position. This offset can be calculated from the raw data as the mean value from the position output within the transport delay phase. The first 5 data points are ignored to reject possible peak values:

```
36. % position offset correction, mean value from first data points:
37.         offset1 = mean(output.signals.values(5:startPoint, 1));
38.         % input: motor torque
39.         in1 = input.signals.values(startPoint:end,1) +offset1;
40.         % output: Omni states; phi, phiDot
41.         out1(:, 1) = output.signals.values(startPoint:end, 1) -offset1;
42.         out1(:, 2) = output.signals.values(startPoint:end, 3);
```

Figure 19: `C_Omni_Data_Aqu` code segment to remove a possible offset.

However, if the data is de-trended, which may give better results, the removal of this offset has no effect.

The top level of the used Simulink experiment diagram can be found in Appendix 16. It uses the phantom Omni block described in 2.3.5.

NOTE: When running the experiment for the Z-Axis, a strong movement of the Y-axis can be observed. This is due to the internal design of the Omni. However, this movement has negligible influence on the captured position data of the Z-Axis. For the demo data sets collected, the Y-axis arm was hold at its initial position manually.

3.2.2 Definition of the Model Structure

The Phantom Omni underlies a strong dry friction influence. This leads to the conclusion, that a linear model will not be able to represent the dynamical behavior of the device very accurate.

As in 3.1.2, the idea is to treat the axes independently for the same reasons. Without a clear knowledge of the internal design of the Omni device, the following general model of a robot arm was used for each axis to describe the physical constrains:

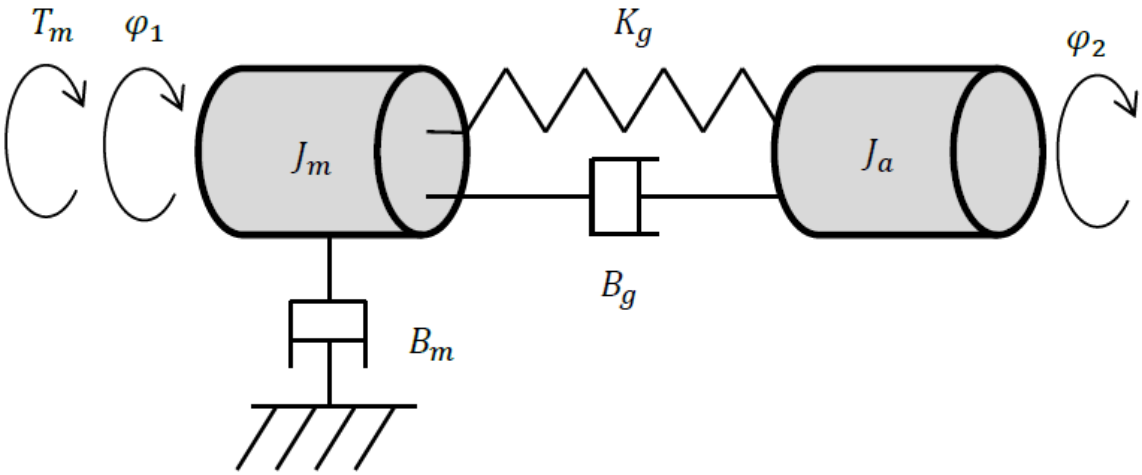


Figure 20: Basic robot arm, physical model.

Were:

Symbol	Description	Unit
J_m	Motor moment of inertia	Kg.m2
J_a	Robot arm moment of inertia	Kg.m2
B_a	Gear-box damping coefficient	Nm.s/rad
K_a	Gear-box stiffness coefficient	Nm/rad
B_m	Motor friction damping coefficient	Nm.s/rad
T_m	Applied motor torque	Nm
phi_1	Motor position	rad
phi_2	Robot arm position	rad

Table 3: Linear robot arm model, physical parameters.

Linear approach

This physical description yields to the following motional differential equations for the two Bodies:

$$\begin{matrix} J_m\ddot{\varphi}_1 + (B_m{+}B_g)\,\dot{\varphi}_1 + K_g\varphi_1 & -B_g\,\dot{\varphi}_2 - K_g\varphi_2 & = T_m \\ -B_g\,\dot{\varphi}_1 - K_g\varphi_1 & +J_m\ddot{\varphi}_2 + B_g\dot{\varphi}_2 + K_g\varphi_2 & = 0 \end{matrix} \qquad (14)$$

In order to obtain the state-space representation, the state, input and output vector is defined as follows:

$$\vec{x}^T = [\varphi_1 \; \varphi_2 \; \dot{\varphi}_1 \; \dot{\varphi}_2 \;] \quad (15)$$

$$\vec{u} = [T_m] \quad (16)$$

$$\vec{y}^T = [\varphi_2 \; \dot{\varphi}_2 \;] \quad (17)$$

And the resulting state-space representation for one axis is:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_g}{J_m} & \frac{K_g}{J_m} & -\frac{(B_m - B_g)}{J_m} & \frac{B_g}{J_m} \\ \frac{K_g}{J_a} & -\frac{K_g}{J_a} & \frac{B_g}{J_a} & -\frac{B_g}{J_a} \end{bmatrix} \quad (18)$$

$$B^T = [0 \quad 0 \quad 1 \quad 0] \quad (19)$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (20)$$

$$D^T = [0 \quad 0 \quad 0 \quad 0 \;] \quad (21)$$

The structure mentioned above was implemented in the function `C_Omni_model_2DOF.c` for the grey box modelling process. This function represents the model in linear form with the first 5 first parameters listed in Table 3.

Nonlinear approach

To improve the model performance, a more advanced description for the friction must be used. As described in [4], the nonlinear friction torque on the motor shaft can be modelled as:

$$T_f(\dot{\varphi}_1) = F_v\dot{\varphi}_1 + (F_c + F_{cs}\,\mathrm{sech}(\alpha\dot{\varphi}_1)) \cdot \tanh(\beta\dot{\varphi}_1) \qquad (22)$$

With

Symbol	Description
F_v	Viscous friction coefficient.
F_c	Coulomb friction coefficient.
F_{cs}	Used to model the Strieback effect.
α	
β	Used to get a smooth model without discontinuity at zero velocity, which is more suitable for simulation

Table 4: Nonlinear friction model, coefficients.

By introducing this friction torque T_f, the linear friction coefficient B_m can be replaced and the motional differential equations turn into:

$$\begin{matrix} J_m\ddot{\varphi}_1 + B_g\,\dot{\varphi}_1 + K_g\varphi_1 & -B_g\,\dot{\varphi}_2 - K_g\varphi_2 & = T_m - T_f \\ -B_g\,\dot{\varphi}_1 - K_g\varphi_1 & +J_m\ddot{\varphi}_2 + B_g\dot{\varphi}_2 + K_g\varphi_2 & = 0 \end{matrix} \qquad (23)$$

The nonlinear derivative state equations are now:

$$\dot{\vec{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{K_g}{J_m} & \frac{K_g}{J_m} & -\frac{B_g}{J_m} & \frac{B_g}{J_m} \\ \frac{K_g}{J_a} & -\frac{K_g}{J_a} & \frac{B_g}{J_a} & -\frac{B_g}{J_a} \end{bmatrix} \vec{x} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \vec{u} + \begin{bmatrix} 0 \\ 0 \\ -T_f \\ 0 \end{bmatrix} \qquad (24)$$

The output equation (17) remains untouched.

This structure with 9 parameters was implemented in the function `C_Omni_nl_model_2DOF_c.c`. The estimation procedure for the nonlinear model structure remains the same as for the linear model, only the parameter needs to be updated.

3.2.3 Parameter estimation

With the more complex, nonlinear model structure and the fact that the initial values are unknown, a more advanced estimation algorithm must be used to obtain usable results. Even if the model structure gives a good description of the real plant, without the correct parameter values the performance will be bad. Furthermore, wrong initial guessed parameters lead the estimation process in the wrong direction, it will take a lot more time and no useful results can be expected. To overcome this problems, the used algorithm is to start with the linear model with only 5 parameters and a *iddata* object with captured values, were the nonlinear behavior is not so important. The estimated values from this linear

model can then be used as initial guess for the nonlinear estimation process were 9 parameters can be estimated.

To perform the estimation process, open the script `C_Omni_Model_Est_main`.

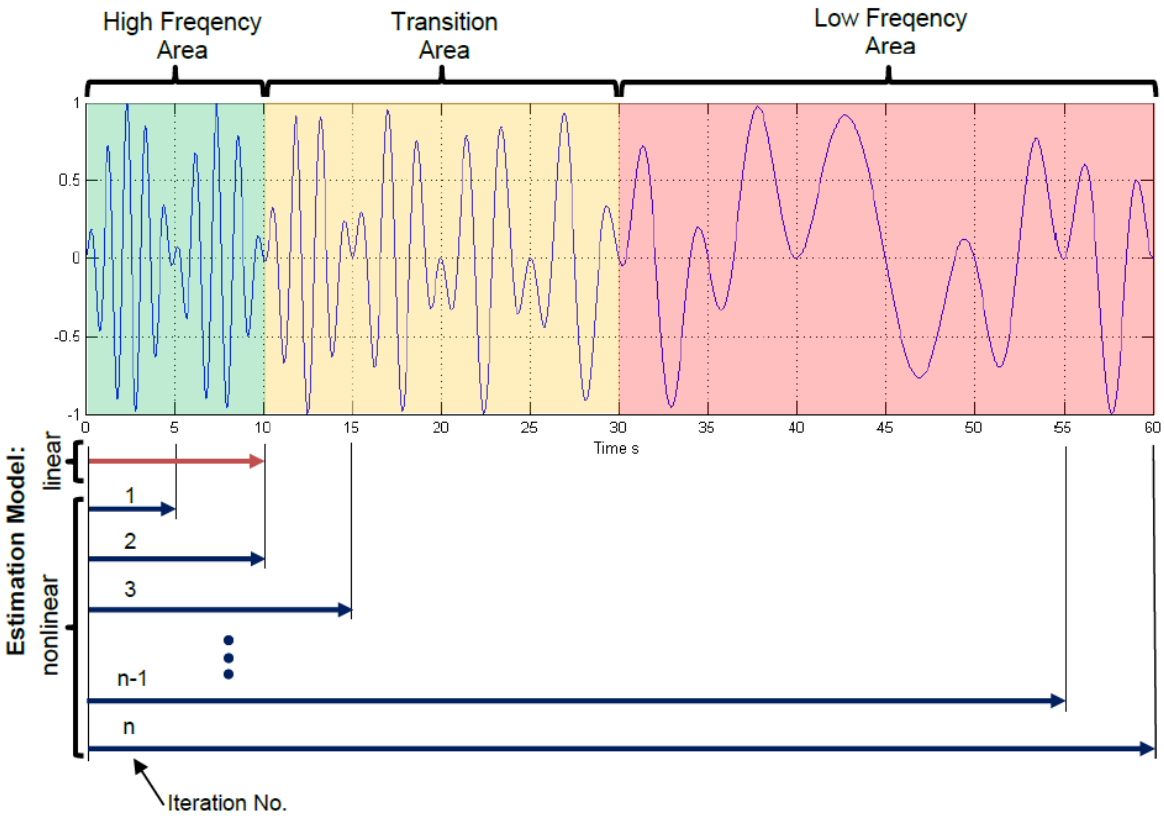


Figure 21: Estimation process algorithm.

The figure above shows an example of a possible data set going from high lo low frequency over time. Assuming a sampling time of 0.05s, the dataset will contain 1200 data points. The estimation process takes now the following steps:

1. Linear model estimation using the first 200 data points only. These are within the high frequency area of the data set (red arrow).
2. Nonlinear model estimation, executed in a loop which takes with each iteration more data points into account (e.g. first iteration data points 1 – 100, second 1-200 etc.). Each iteration takes more data points into account (blue arrows) and takes the estimated parameters from the previous iteration as start parameters.

The data set should start with high amplitude high frequency movements as the dry friction is not significant here. Within the data set, the frequency can then be lowered which yields to slower movements were the nonlinear friction must be taken into account.

NOTE: Decreasing the max. iteration steps (`MAX_ITER`) does not always mean the estimation process will be faster, the opposite may be true. This is the case if one data set part needs more iterations inside the `pem` function to achieve good results. If these results are inaccurate, the next iteration starts with wrong initial parameters which leads to a long estimation process with no useful results.
As rule of thumb can be said, if one iteration within `pem` takes more than 5-10 sec, try to restart with different settings. To stop the execution press *Ctrl +c* in the command window.

3.2.4 Model Verification

The following figure shows the performance of the nonlinear model compared to the linear model if both were estimated with all data points of an *iddata* object. The estimated parameter for these models can be found in Appendix 8. As the position depends on the velocity, it is treated as secondary information about the model quality. More important is the velocity behaviour of the model compare to the measured data.

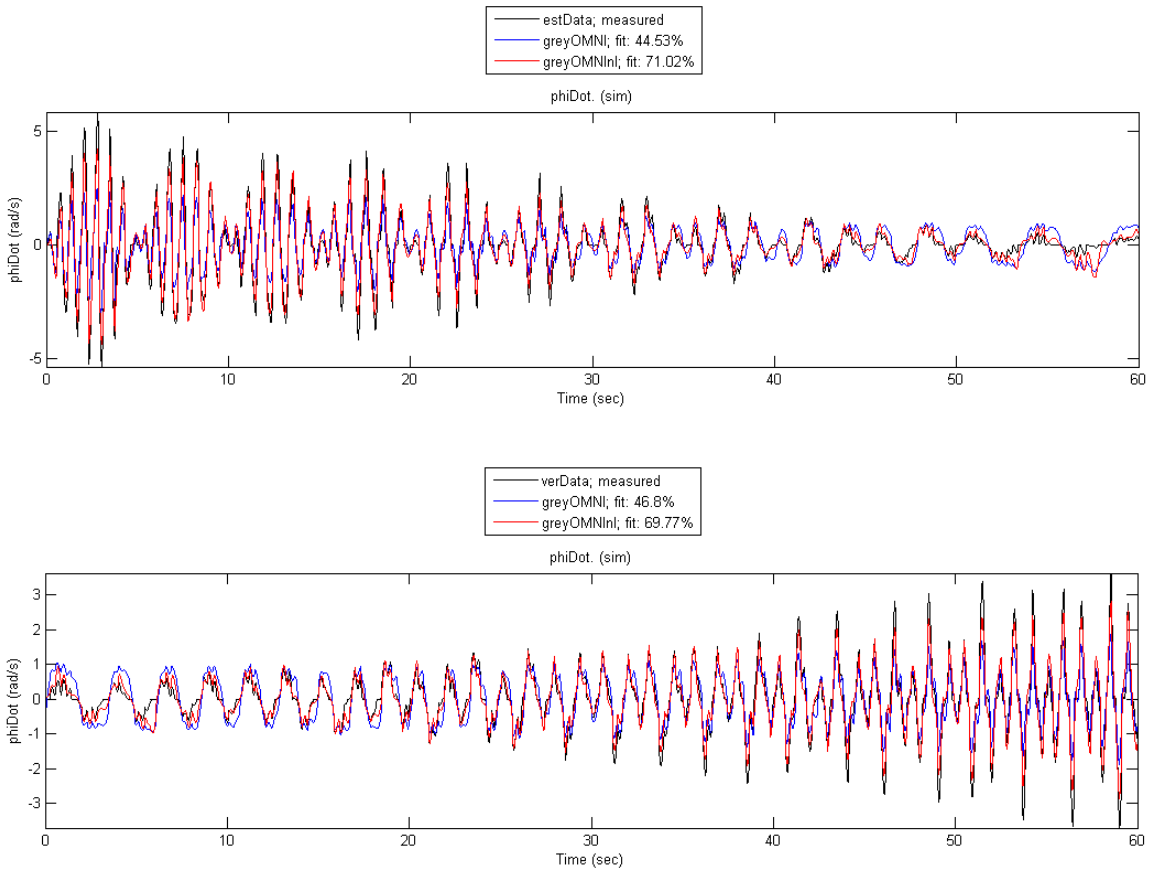


Figure 22: Y-Axis model verification, velocity comparison with estimation data (upper plot) and verification data (lower plot).

It can be observed that the linear model `greyOMNI` gives not a bad representation (44.5-46.8% fit), even at the lower frequencies. However, at the lower frequencies the amplitude is too high and at the higher frequencies too low. The model dynamic is too limited to give a better representation. The nonlinear model `greyOMNInl` performs far better over the whole frequency range (69.8-71% fit).

One demo model (*yAxisDemoModel* and *zAxisDemoModel)* for each axis was estimated and saved in the file `C_Omni_ModelObjects.mat`. Their parameter values and verification plots can be found in Appendix 9 and Appendix 10.

The models were estimated with the *iddata* objects *yAxisDemoData* respective *zAxisDemoData*. They were verified with the *iddata* objects *yAxisDemoData2* and *zAxisDemoData2*. The fit compared to the velocity signal of the verification data is:

- 68.47% for the Y-Axis
- 63.34% for the Z-Axis

4 SCENARIO 1: MASTER PHANTOM OMNI, SLAVE 2DSFJ ROBOT

In the previous chapter, model structures of the real plant were defined, their parameters estimated and their performance verified. In this and the next chapter, these models are now used to implement simulation diagrams which represent the plants in the two scenarios. Each scenario contains one Simulink representation for the simulation and one for the real plant experiment.

4.1 Operation Description

In this first scenario, the Phantom Omni haptic device controls the 2DSFJ robot. The Y-Axis controls Stage1 and the Z-Axis Stage2. Two major situations can be distinguished, the movement in free space and the contact with a hard surface.

As the Omni does not enclose a force measurement sensor, a fictitious spring is used to get a force measurement according to the position.

Movement in free space:

The operator moves the Omni in a desired direction, as farer away from the neutral position as more force must be applied due to a fictitious spring. The slave reacts to this commanded torque with a movement in the same direction at a speed proportional to the applied force. (Theoretically) no torque is applied from the environment to the slave, the operator feels no additional resistance to his commands.

Control mode: Torque → velocity, closed loop

In contact with a hard surface

When the slave touches a hard surface, its arm movement will be stopped. The torque applied from the robot to the environment can be measured by the joint deflection. This torque is feed back to the master with a proportional factor. The operator feels an additional resistance on the master device.

Control mode: Torque, closed loop

The following table shows all MATLAB files associated with this section:

File Name	Description	Location Folder
constants.m	Contains general constants definitions, Appendix 27.	0_Parameters
robot_2DSFJ_setup.m	Basic parameters needed for the real plant, Appendix 28.	0_Parameters
phantomOmni_setup.m	Basic parameters needed for the real plant, Appendix 29.	0_Parameters
SignalGeneratorGUI.m	Function for the signal generator operation, Appendix 30.	0_Parameters
A_Scen1_main.m	Real plant setup, main script.	1_Scenario1
AB_Scen1_main.m	Simulation OR real plant setup, main script, Appendix 38.	1_Scenario1
B_Scen1_main.m	Simulation setup, main script.	1_Scenario1
A_Scen1_RealPlant.mdl	Simulink diagram, real plant	1_Scenario1
B_Scen1_Sim.mdl	Simulink diagram, simulation.	1_Scenario1
D_Scen1_Exp_Analysis.m	Script to analyse the captured data, Appendix 39.	1_Scenario1
Scen1_createfigure.m	Function containing figure formatting commands, used to plot simulation data.	1_Scenario1
C_2DSFJ_ModelObjects.mat	Contains the models objects.	3_2DSFJ_Model_Est

Table 5: MATLAB files used within the first scenario.

4.2 Simulation

Figure 23 shows the basic layout used for both scenarios. The simulation and the real plant diagrams have exactly the same structure. A controller designed with the simulation setup can be included into the real plants setup simply by copy-paste. All input/output signals to and from the controller block are the same.

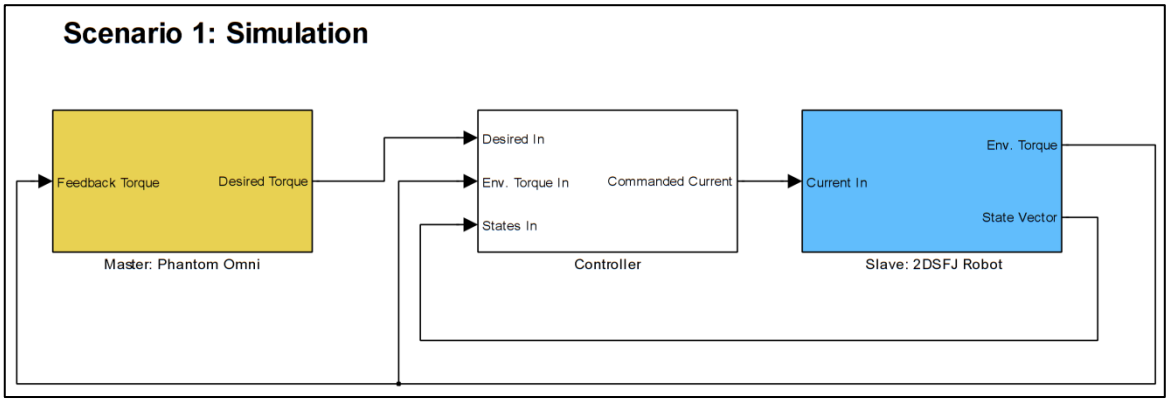


Figure 23: Scenario 1 Simulation top level, the Simulink diagram structure for both scenarios consists of 3 main blocks: Master, Controller and Slave.

4.2.1 Master

The position movements on the master are simulated by the signal generator. The use of a model for the Omni has no influence to the simulation results and is therefore not necessary. However, the feedback loop is in place to comply with the real plant.

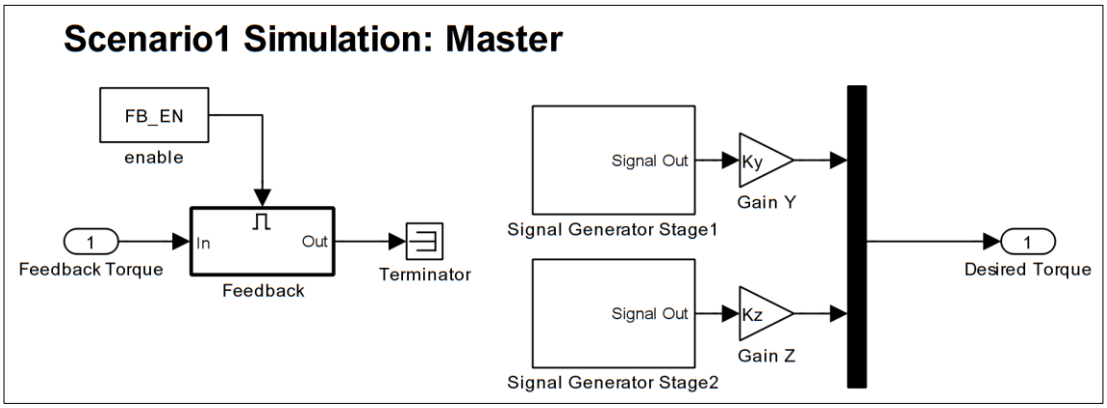


Figure 24: Scenario 1 Simulation Master block, two instances of the signal generator block are used to simulate the position movements of both axis. A gain block converts the position to a desired torque signals.

4.2.2 Slave

The slave blocks top level contains blocks for both robot stages (Appendix 17). Each stage (Figure 25) contains a nonlinear Grey-Box model block. The model used must be set in one of the scripts containing the simulation settings. The structure of the slave offers the possibility to simulate a hard surface with a saturation block, this block limits the movements of the model within a user defined range (`ST1_WALL`). A derivative filter must be used to get the velocity signal after the saturation.

NOTE: The wall contact saturation block should be used only on an informative base. The position within the model will not be affected by this block and the model output after returning from the wall contact may be inaccurate. So it may only be representative for the first wall contact.

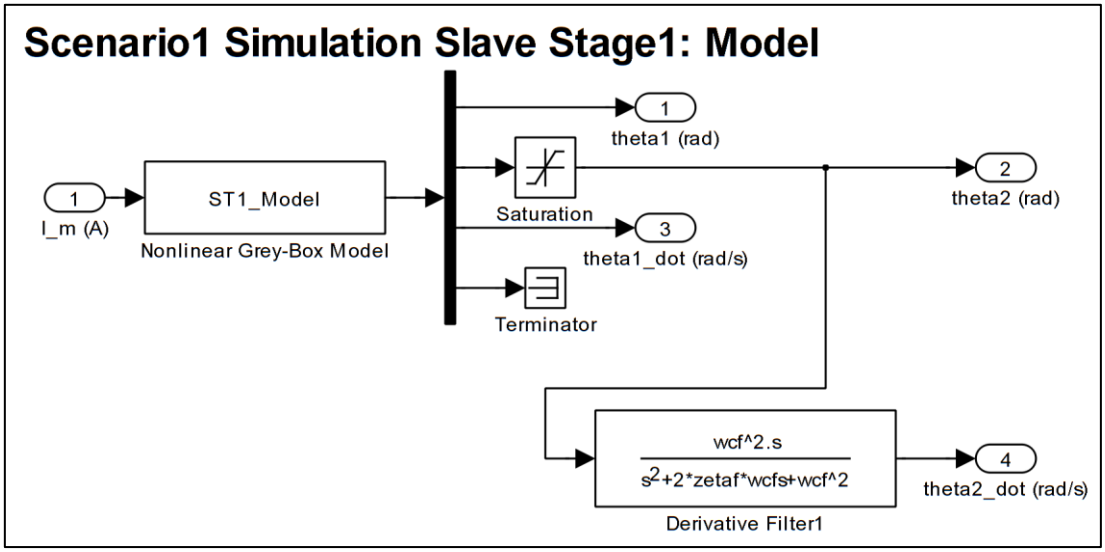


Figure 25: Scenario 1 Simulation Slave, a saturation block permits the simulation of a contact with a hard surface.

4.2.3 Controller

The controller design follows the force control principle presented in [1] in a simpler PID-design. The basic idea is to have a 2 channel force control setup, which means to have a torque command from the master and a torque feedback from the slave. The controller should be able to control the slave in free space and while in contact with a hard surface without the need of changing the controller design or parameter values.

The torque error (25) is defined as the difference between the environmental torque T_e and the applied, desired human torque T_d on the master:

$$\epsilon = T_e - T_d \qquad (25)$$

The environment torque T_e is defined as the torque applied from the robot to the environment:

$$T_e = K_s(\theta_1 - \theta_2) \qquad (26)$$

Where K_s is the joint torsional stiffness. Ideally, in free space θ_1 is equal to θ_2 and the environmental torque becomes 0.

NOTE: To comply with the definitions in 2.4.1, a positive T_e results if the robot arm moves in CCW direction and hits a stiff surface. In this case θ_1 will be greater than θ_2.

Control in free space

To control the robot in free space, a velocity penalty is added to the torque error which leads to the augmented error defined as:

$$s = \Lambda\epsilon + \dot{\theta}_1 \qquad (27)$$

With Λ as a scale factor and $\dot{\theta}_1$ as the motor speed of the 2DSFJ robot. If the controller can achieve s = 0 in free space were T_e is 0, (27) will become to:

$$\dot{\theta}_1 = \Lambda T_d \qquad (28)$$

The slave velocity will be proportional to the desired torque.

Control in contact with hard surface

If T_e is not 0, the commanded torque T_c is equal to T_e plus the PI-controlled augmented error:

$$T_c = T_e + K_{p1} \cdot s + K_i \int s \qquad (29)$$

Which means T_e will be equal to T_c if the controller can achieve s = 0. The commanded torque depends on the difference between the actual position of the arm and the desired position of the motor multiplied with the joint torsional stiffness:

$$T_c = K_s(\theta_2 - \theta_{1des}) \qquad (30)$$

Which leads to the desired motor position:

$$\theta_{1des} = \theta_2 - \frac{1}{K_s} T_c \qquad (31)$$

By using the desired position, the actual position and the actual velocity of the motor, the commanded motor current will be:

$$I_m = K_{p2}(\theta_{1des} - \theta_1) - K_d\dot{\theta}_1 \qquad (32)$$

The top level of the Simulink controller block is attached in Appendix 18. The equations above lead to the representation for one stage as shown in Appendix 19. This controller layout has 5 adjustable parameters for each stage. After trial and error, the values listed in the tables of Appendix 11 were found for the two stages.

4.2.4 Run the simulation

To run the simulation, open first the script `B_Scen1_main` (or `AB_Scen1_main`. and uncomment the simulation diagram)

After the loading section, the simulation setup section allows altering the basic settings like simulation duration, human torque gains as well as the model used within the simulation. The following sections contain the controller parameters, the default signal generator and feedback settings.

By executing the script, the Simulink diagram and the signal generator GUI open. The simulation diagram itself can then be executed as usual inside Simulink.

4.3 Real Plant

The real plant experiment (Figure 26) for the first scenario uses both robots. The experiment can be run in manual mode which means the master robot is actuated by a human operator or in simulated mode meaning the position of the master depends on a generated position pattern. The feedback loop permits the haptic control experience with scalable gain.

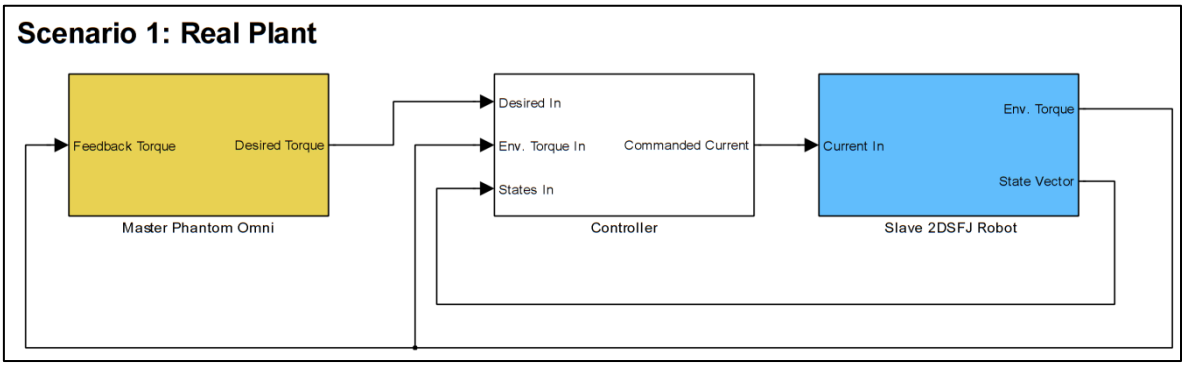


Figure 26: Scenario 1 Real Plant top level.

4.3.1 Master

The master block (Simulink diagram in Appendix 20) contains the Phantom Omni block described in 2.3.5 with some minor changes:

- The velocity signal isn't used.
- The neutral position is maintained by fictitious springs (Figure 27). As farer away from the neutral position, as more force must be applied by the operator. The fictitious spring gain and the desired torque gain factor can be adjusted independently.

- In order to compare the behaviour of the controller to the simulation, the real plant offers on option to disable the manual actuation of the master device and use the signal generator instead (Figure 28).

  NOTE: The master is meant to be held by the operator, especially with enabled feedback. When using the real plant experiment in simulation mode, it is strongly recommended to disable the feedback loop.

- Feedback block (Figure 29), the gain of the feedback is adjustable and the feedback signal may be filtered. By filtering the feedback signal, the systems stability is improved when using higher feedback gains.

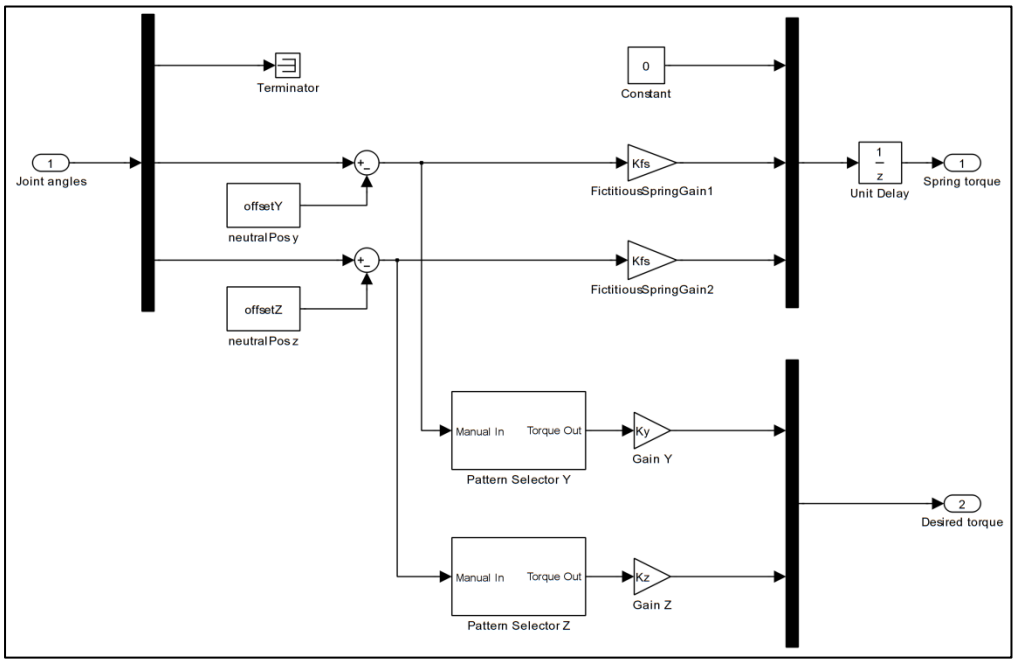


Figure 27: Position to torque converter block, calculates the spring torque and the desired torque out of the actual position of the device.

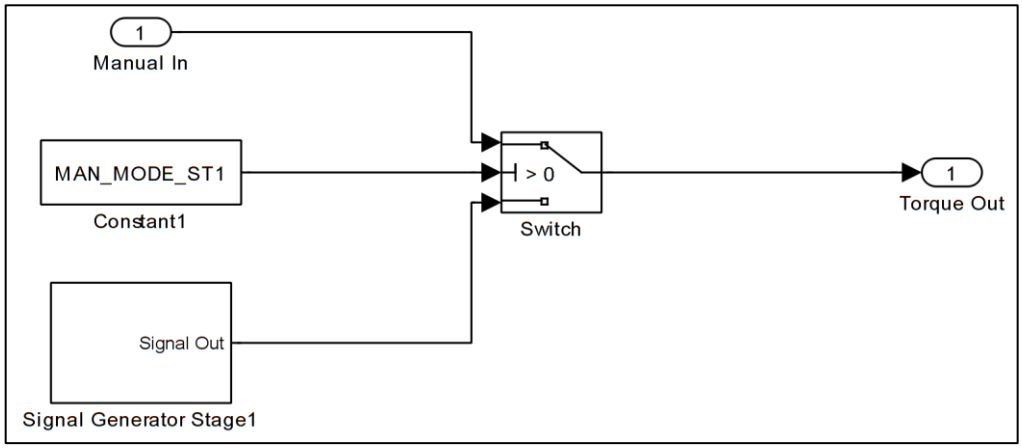


Figure 28: Pattern selector block, manual or generated position patterns can be used.

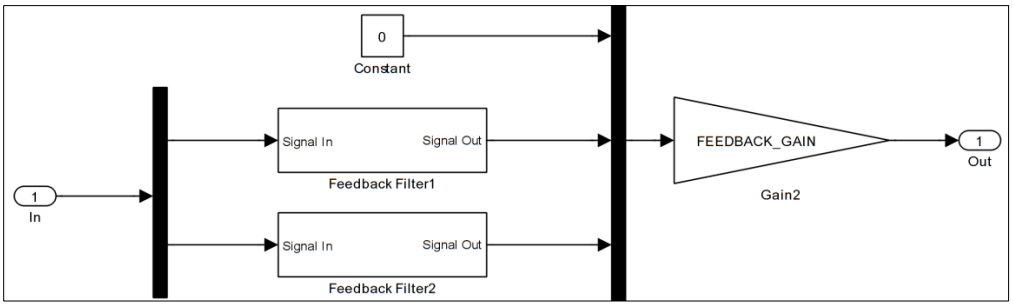


Figure 29: Feedback block, allows the feedback torque to be filtered.

4.3.2 Slave

The slave top level design is the same as in shown in Appendix 17, but each stage (Figure 30) contains now a 2DSFJ block described in 2.4.4.

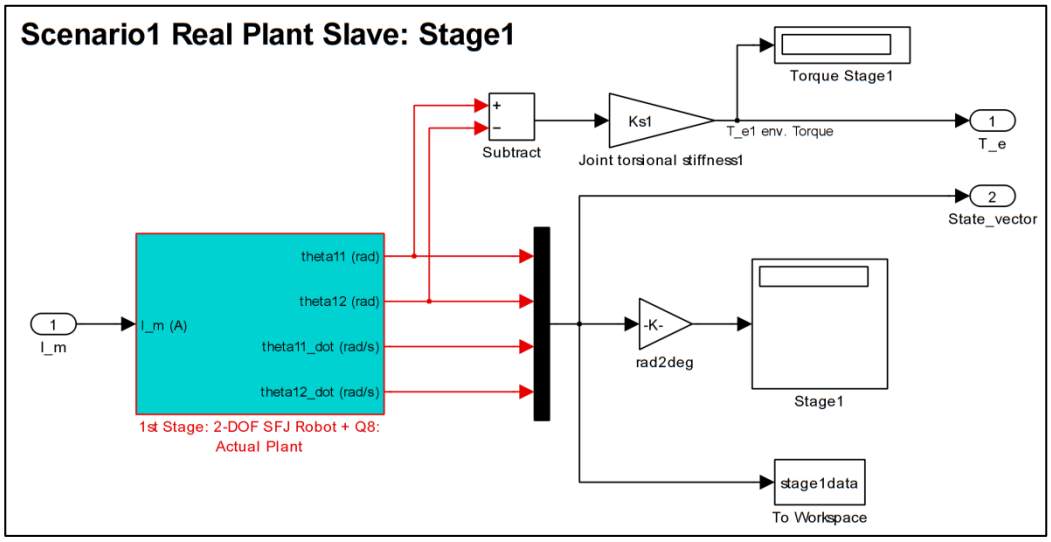


Figure 30: Scenario 1 salve, implementation of Stage1. The environmental torque defined in 4.2.3 is measured inside this block, it is used for the controller and as feedback for the master.

4.3.3 Controller

The controller block used in the real plant experiment is identical with the one used in the simulation (see 4.2.3).

4.3.4 Run the Experiment

To run the experiment, open the script `A_Scen1_main` (or `AB_Scen1_main`. and uncomment the real plant diagram). A rebuild of the Simulink diagram may be necessary before connecting to the target.

Executing this script opens the real plant Simulink diagram and if the manual mode is disabled, the signal generator GUI.

IMPORTANT NOTE: Before clicking on RUN in Simulink, be sure to put the Omni horizontally and hold the arms near to their neutral positions. This is also necessary when the manual mode is disabled, the Omni will be powered anyway!

Start the experiment inside Simulink.

4.4 Verification

To verify the performance of the designed controller as well as the behaviour of the model compared to the real plant, both, the simulation and the real plant Simulink diagrams can be actuated with the same generated input signal. To compare the output of this experiment, the script `D_Scen1_Exp_Analysis` can be used, this script plots the results in formatted form. The signal delay may be removed or not.

NOTE: The data collected from a real plant experiment were not saved directly into the workspace, they were stored in a *`realPlantExperimentName`*`.mat` file in the current open folder. This is due to the external running mode within Simulink. Each time the real plant experiment is run, this file is overwritten. The values from the simulation were directly saved into the workspace.

To prevent overwriting the variables in the workspace before plotting, the analysis script should be run after each experiment or simulation. The simulation and experiment plots are displayed always in a new window. If the collected data needs to be available later, it must be saved manually.

4.4.1 Results: Simulation mode

Actuation signal:

The performance of the stages was verified using the following values for the generated position pattern:

Stage 1

Step response, amplitude 0.06 rad, torque gain 10Nm/rad, lambda 0.5, wall contact at 1 rad.

Stage 2

Step response, amplitude 0.08 rad, torque gain 10Nm/rad, lambda 0.5, wall contact at 1 rad.

The following plots show the simulation and the real plant experiment output for both stages:

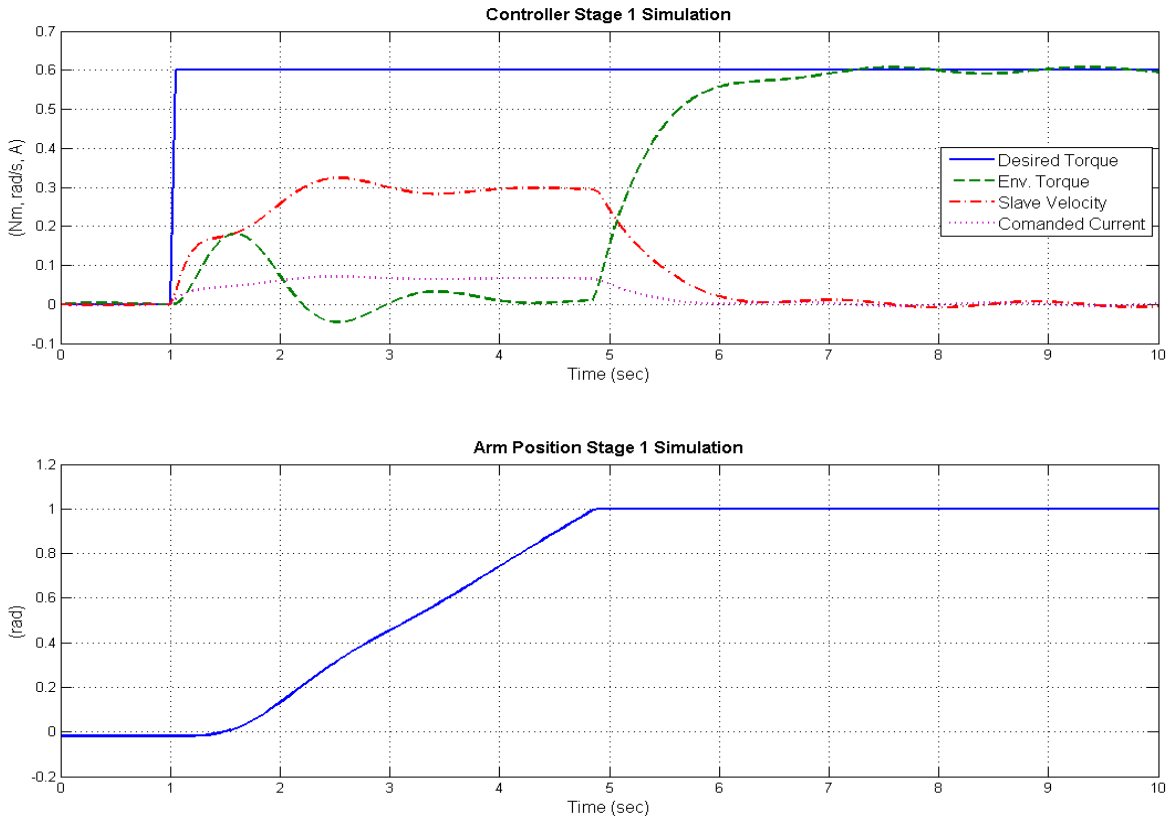


Figure 31: Scenario 1 verification, Stage 1 simulation plot. Used model: `st1DemoModel`.

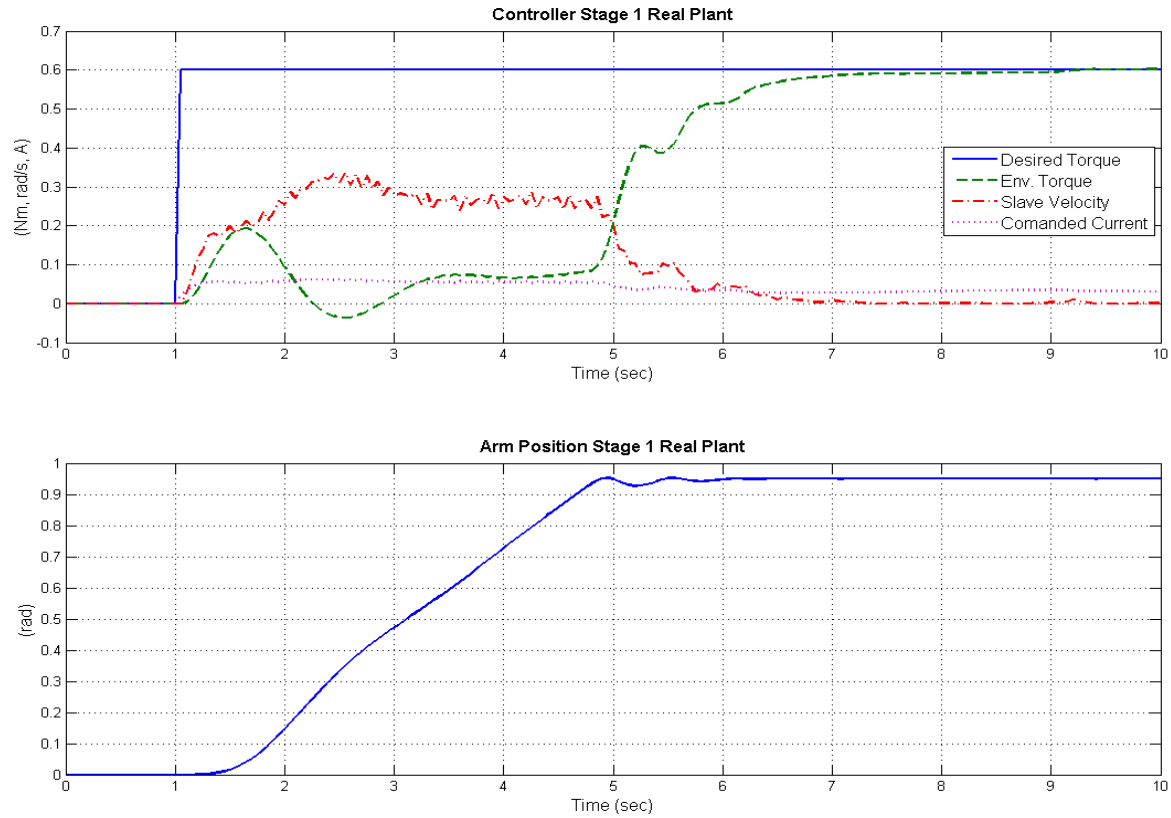


Figure 32: Scenario 1 verification, Stage 1 real plant experiment plot.

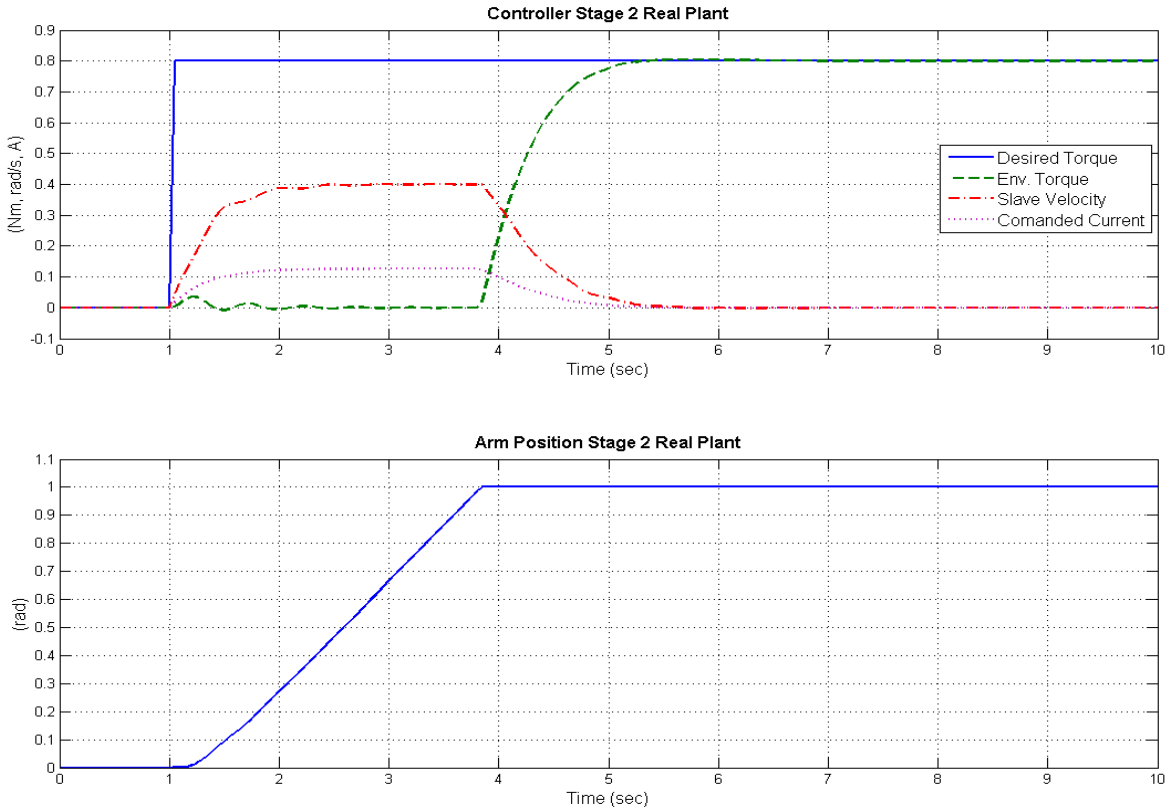


Figure 33: Scenario 1 verification, Stage 2 simulation plot. Used model: `st2DemoModel`

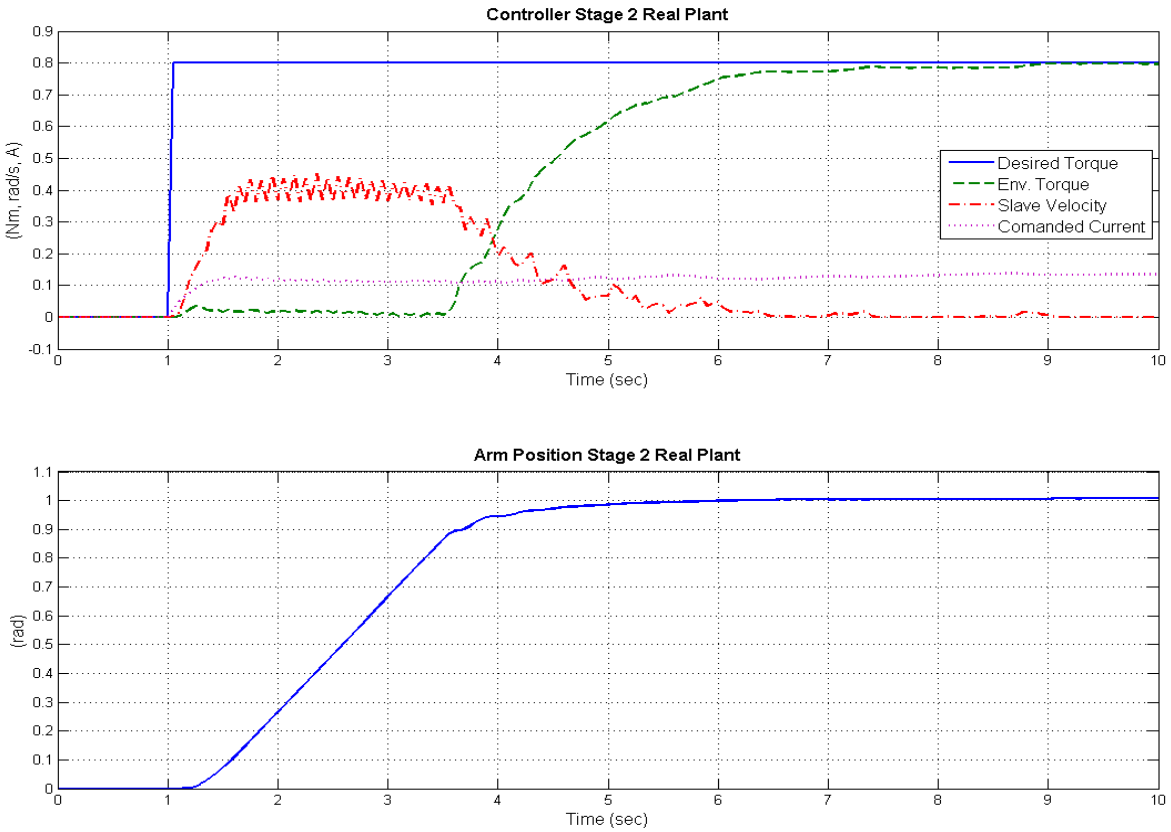


Figure 34: Scenario 1 verification, Stage 2 real plant experiment plot.

The resulting plots (Figure 31 - Figure 34) can be divided into 3 regions.

Free Space[6]

By locking at the first seconds after the step time, the following statements can be made for both stages:

- The velocity signal from the real plant is subject to disturbances.
- The velocity and environment torque signal from the simulation and the real plant experiment have a similar shape.
- The velocity signal's amplitude is not exactly 0.5 of the desired torque. This error is due to the environment torque which is not equal to 0.

Transition

By looking at the transition area from movement to full stop, the following observation can be made:

- The transition happens smoothly, the motor velocity decreases as and the applied environment torque increases without peaks or high oscillations.
- The simulation permits to have a non-elastic hit to a hard surface, the kinetic energy is instantly absorbed due to the use of a saturation block. Even though, the motor velocity is reducing and the applied torque is building up smoothly.
- The transition on Stage2 takes place approx. 50% faster in the simulation.

Contact with hard surface

After the transition the contact with the hard surface is maintained:

- Simulation and real plant experiment achieve zero torque error on both stages.
- No motor current is applied to maintain the environment torque in the simulation. This is due to the lack of the force from the environment to the robot arm.

4.4.2 Results: Manual mode

While operating the system in manual mode, the operator can make the following statements about the systems behaviour:

- The control movements are intuitive, the slave response fast.
- The system behavior follows the expectations from the simulation.
- The feedback permits a haptic experience: When touching a hard surface, the commanded torque is reduced immediately.
- While controlling the robot in free space, switching the movement direction causes a haptic feedback as well.
- The Omni put in horizontal mode is not stiff enough, it slips over the table while feedback torque is applied if the case is not held by the other hand.

[6] Free space in this context means not in contact with an obstacle

5 SCENARIO 2: MASTER 2DSFJ ROBOT, SLAVE PHANTOM OMNI

This chapter treats the second scenario, as in chapter 4, two Simulink diagrams were implemented. One for the simulation and one for the real plant experiment.

5.1 Operation Description

In this scenario, the 2DSFJ robot is the master and the Phantom Omni is the slave. Stage1 controls the Y-Axis and Stage2 the Z-Axis. Two major situations can be distinguished, the movement in free space and while in contact with a hard surface.

Movement in free space:

The operator moves the robot arm in a desired direction, at a desired velocity. The slave follows this movement. At the same time, the position of the slave is reflected back to the master, were the motors of each stage follows the movements of the slave. Ideally, no torque will be applied on the master, the operator feels almost no resistance to his commands.

Control mode: Position, closed loop

In contact with a hard surface

When the slave touches a hard surface, its arm movement will be stopped. This leads to a stopped motor on the master at the position, were the contact occurred. Due to the joint torsional stiffness, the operator will feel now a resistance against his commands. As farer away he moves the arm from the contact position, as higher is the torque the operator must apply. This torque can be measured as the commanded torque. The salve applies a torque proportional to the commanded torque to the contact surface.

Control mode: Torque, open loop

The following table shows all MATLAB files associated with this chapter:

File Name	Description	Location Folder
constants.m	Contains general constants definitions, Appendix 27.	0_Parameters
robot_2DSFJ_setup.m	Basic parameters needed for the real plant, Appendix 28.	0_Parameters
phantomOmni_setup.m	Basic parameters needed for the real plant, Appendix 29.	0_Parameters
SignalGeneratorGUI.m	Function for the signal generator operation, Appendix 30.	0_Parameters
A_Scen2_main.m	Real plant setup main script	2_Scenario2
AB_Scen2_main.m	Simulation AND real plant setup main script, Appendix 40.	2_Scenario2
B_Scen2_main.m	Simulation setup main script	2_Scenario2
A_Scen2_RealPlant.mdl	Simulink diagram, real plant	2_Scenario2
B_Scen2_Sim.mdl	Simulink diagram, simulation	2_Scenario2
D_Scen2_Exp_Analysis.m	Analyse the captured experiment data, Appendix 41.	2_Scenario2
Scen2_createfigure.m	Function containing figure formatting commands, used to plot simulation data	2_Scenario2

Table 6: MATLAB files associated with the second scenario.

5.2 Simulation

This scenario uses position reflection back to the master, therefore models for both, the master and the slave must be used within the simulation diagrams. The basic layout (Figure 35) follows the design of scenario 1 with three main blocks on the top level.

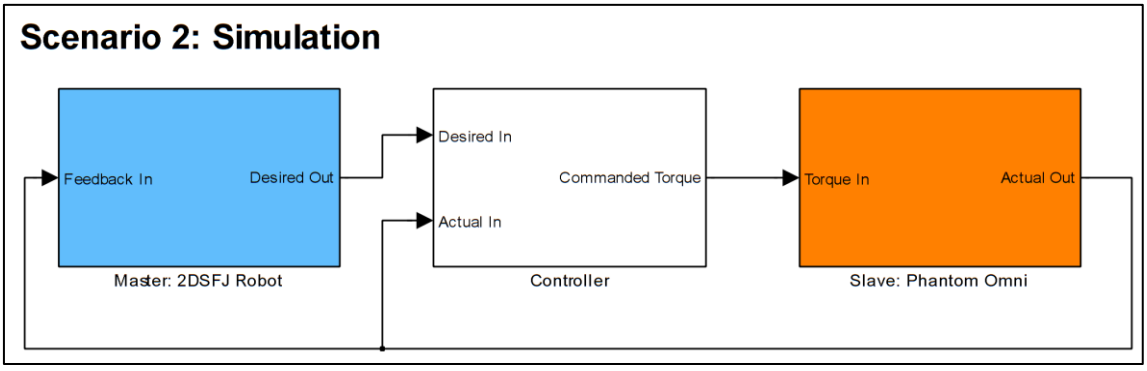


Figure 35: Scenario 2 Simulation, top level.

5.2.1 Master

The master block top level contains the two stages (Appendix 21). Each stage can be disabled which means that also the reflection to this stage will be disabled. The position reflection for each stage (Appendix 22) has its own PID controller.

The values for the 3 reflection controller parameters for each stage have been found by trial and error, they are listed in Table 6 and Table7 attached in Appendix 12.

The representation block for one stage itself (Figure 36) consist of a Grey-Box model block, where the used model must be defined. The position movements from the operator are simulated by the signal generator (arm movements). The reflection control signal (motor current) is feed into the model block and leads to position and velocity outputs for the motor drive.

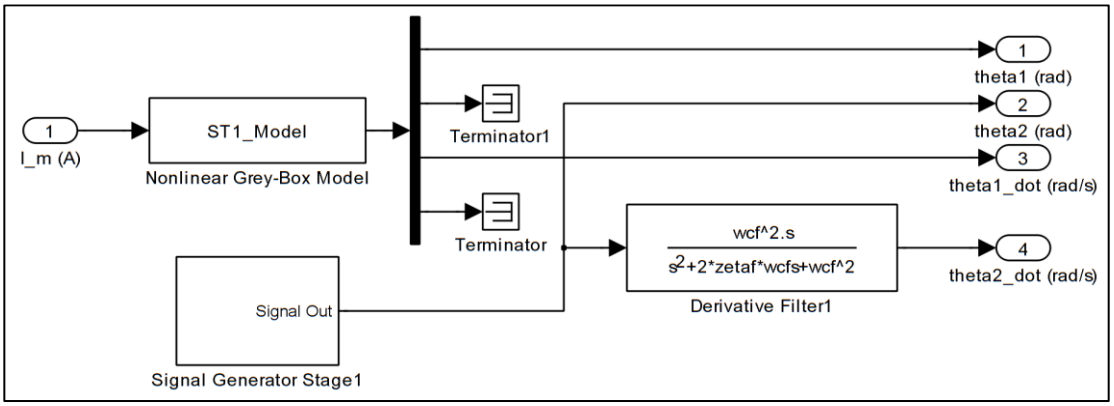


Figure 36: Scenario 2 2DSFJ Robot Model block, Stage1.

5.2.2 Slave

The slave block contains representations for both Omni axes (Figure 37). Each Axis (Figure 38) has the possibility to simulate a hard surface at a given position with a saturation block. Again, a nonlinear Grey-Box model block is used to represent the model structure. The velocity measurement is extracted by a second order low pass filter.

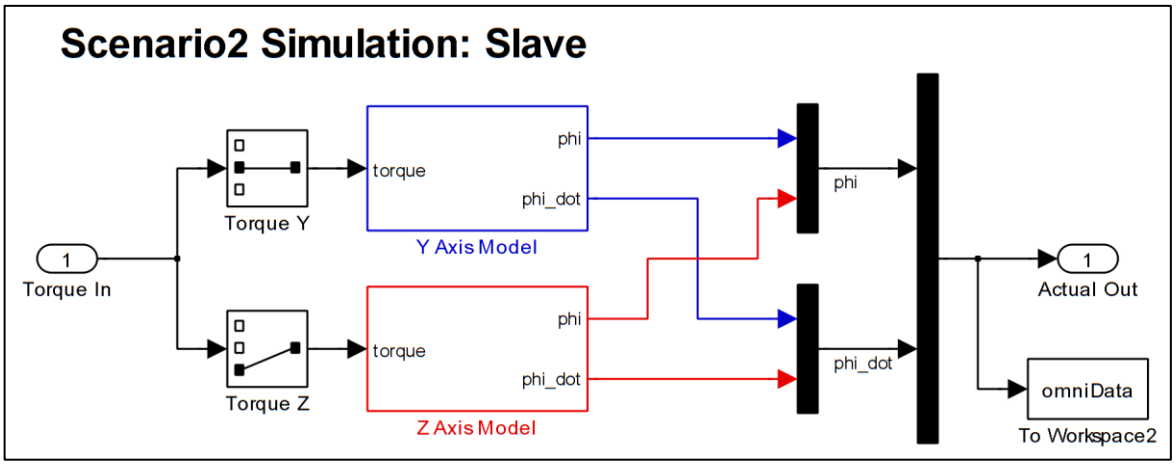


Figure 37: Scenario 2 simulation, slave top level.

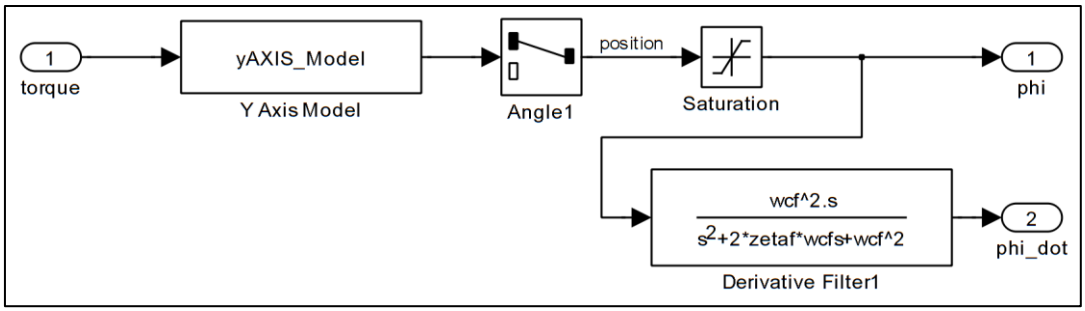


Figure 38: Scenario 2 Simulation, slave block Y-Axis.

NOTE: The wall contact saturation block should be used only on an informative base. The position within the model will not be affected by this block and the model output after returning from the wall contact may be inaccurate. So it may only be representative for the first wall contact.

5.2.3 Controller

The controller must be able to control the position in free space and the torque while in contact with a hard surface. There is no torque measurement installed on the Omni but the input signal is a vector containing the desired torque for each axes. This torque can be read as applied torque from the Omni to the environment. To achieve a desired torque on the slave, it is sufficient to command this torque at the input of the device (open loop). The desired torque on the master is defined as follows:

$$T_d = K_s(\theta_2 - \theta_1) \quad (33)$$

Where K_s is the joint torsional stiffness. With perfect reflection, the motor position θ_1 is equal to the actual position φ.

$$T_d = K_s(\theta_2 - \varphi) \quad (34)$$

While in contact with a hard surface, the commanded torque should be the desired torque with a proportional gain Λ (open loop):

$$T_c = \Lambda \cdot T_d \quad (35)$$

Defining again an augmented error s with Ψ as a constant and $\dot{\varphi}$ as the slave speed:

$$s = \Psi T_d - \dot{\varphi} \quad (36)$$

NOTE: In order to comply with the open loop torque control, the slave velocity has to be subtracted this time.

In free space, the commanded torque depends on the augmented error:

$$T_c = K_p \cdot \mathrm{s} \quad (37)$$

Or:

$$T_c = K_p(\Psi \cdot T_d - \dot{\varphi}) \quad (38)$$

In contact with a hard surface, $\dot{\varphi}$ becomes 0. To obtain the relation (35), Ψ must be:

$$\Psi = \frac{1}{K_p}\,\Lambda \qquad (39)$$

The top level of the controller diagram can be found in Appendix 23. Each stage has its own controller block (Appendix 24). The controller block itself contains the representation of the equations above, its diagram can be found in Appendix 25. Slider gains are present in order to fine-tune the controller if desired.

The values for the 2 controller parameter Λ and K_p for both axes are found by trial and error, they are listed in Table 8 and Table 9 attached in Appendix 12.

5.2.4 Run the simulation

To run the simulation, open first the script `B_Scen2_main` (or `AB_Scen2_main`. and uncomment the simulation diagram)

The basic settings inside these scripts are the same as in mentioned in 4.3.4.

5.3 Real Plant

The real plant experiment uses again both robots, therefore both hardware representations are used. The structure of the top level (Figure 39) is the same as in the simulation. As in scenario1, the master possesses the option to actuate the slave with an artificial input signal provided by the signal generator.

NOTE: Even if the master's position is simulated, when the reflection is enabled both robots will move.

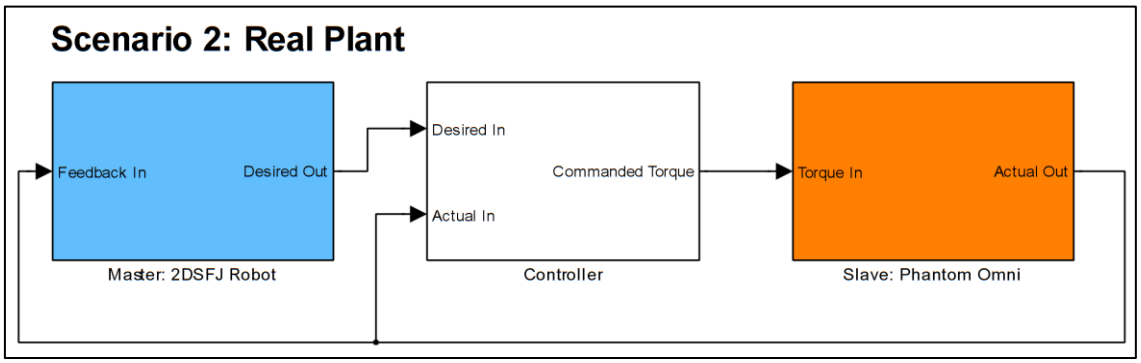


Figure 39: Scenario 2 Real Plant top level.

5.3.1 Master

The layout of the master block is exactly the same as in 5.2.1, but the real plant blocks described in 2.4.4 are used to represent the robot stages. One minor modification has been made inside these blocks to have the possibility to simulate a position pattern (see Appendix 26). The block Manual Mode Switch (Figure 40) has been added to the arm position measurement signal, it permits to disable the manual mode and use a generated position signal instead.

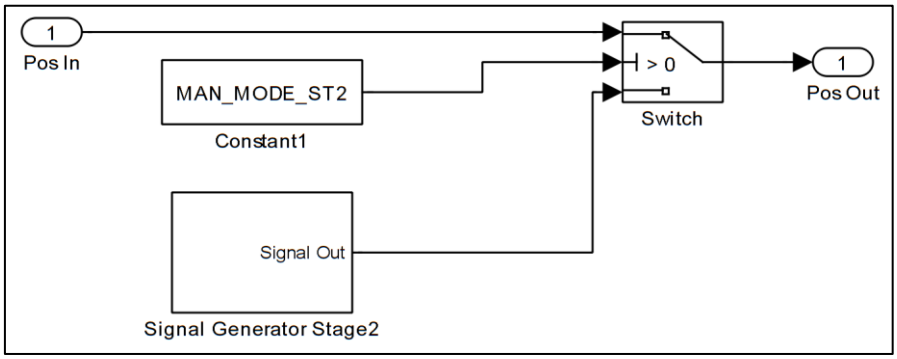


Figure 40: Manual Mode Switch block

5.3.2 Slave

The slave block contains exactly the Simulink diagram described in 2.3.5. A torque limiting saturation block has been added to protect the hardware form excessive load.

5.3.3 Controller

The controller block is absolute identical with the one described in the simulation diagram 5.2.3.

5.3.4 Run the experiment

To run the experiment, open the script `A_Scen2_main` (or `AB_Scen2_main`. and uncomment the real plant diagram). A rebuild of the Simulink diagram may be necessary before connecting to the target.

IMPORTANT NOTE: Before clicking on RUN in Simulink, be sure to put the Omni horizontally and hold the arms close to their neutral positions, even though it is the slave. With enabled feedback, the position of the Omni will be reflected on the 2DSFJ robot. By clicking on RUN, the robot will move instantly to align with the Omni.

5.4 Verification

The verification process is basically the same as in 4.4, the same artificial signal is used to actuate the simulation and the real plant experiment and the output is compared. The script `D_Scen2_Exp_Analysis` can be used to plot the captured experiment or simulation data.

5.4.1 Results: Simulation Mode

Each subplot contains information about:

- Master and slave arm position.
- Slave arm and master motor position (reflection).
- Master and slave arm velocity.
- Desired and commanded torque.

Actuation signals:

The system is not as stable as in scenario 1. Therefore no step signal is used for the verification. To run the system from free space to a hard surface, a sine signal with amplitude 1 and a frequency of 0.025Hz was used. The hard surface is simulated at 0.4 rad. Lambda is set to 0.5 for the Y-Axis and to 0.3 for the Z-Axis.

The following plots show the results for the simulation and the real plant experiment for both stages:

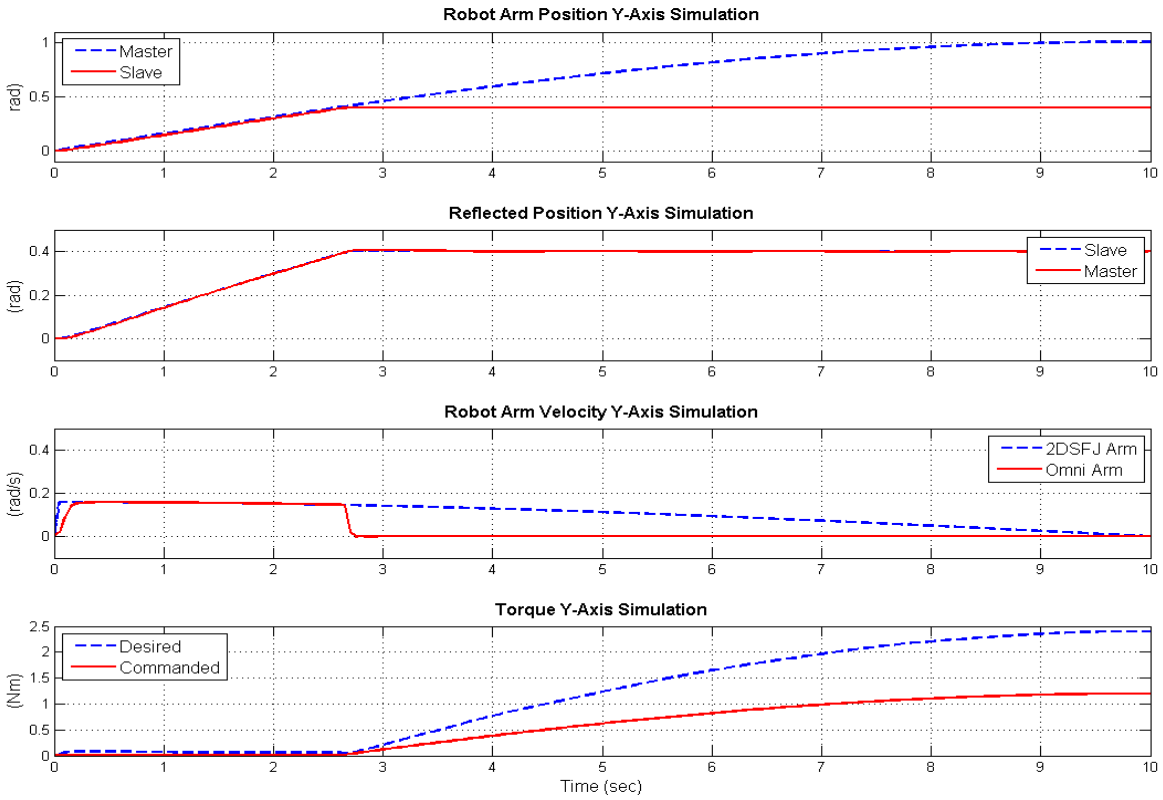


Figure 41: Scenario. 2 verification, Y-Axis simulation plot. Used model: `yAxisDemoModel`.

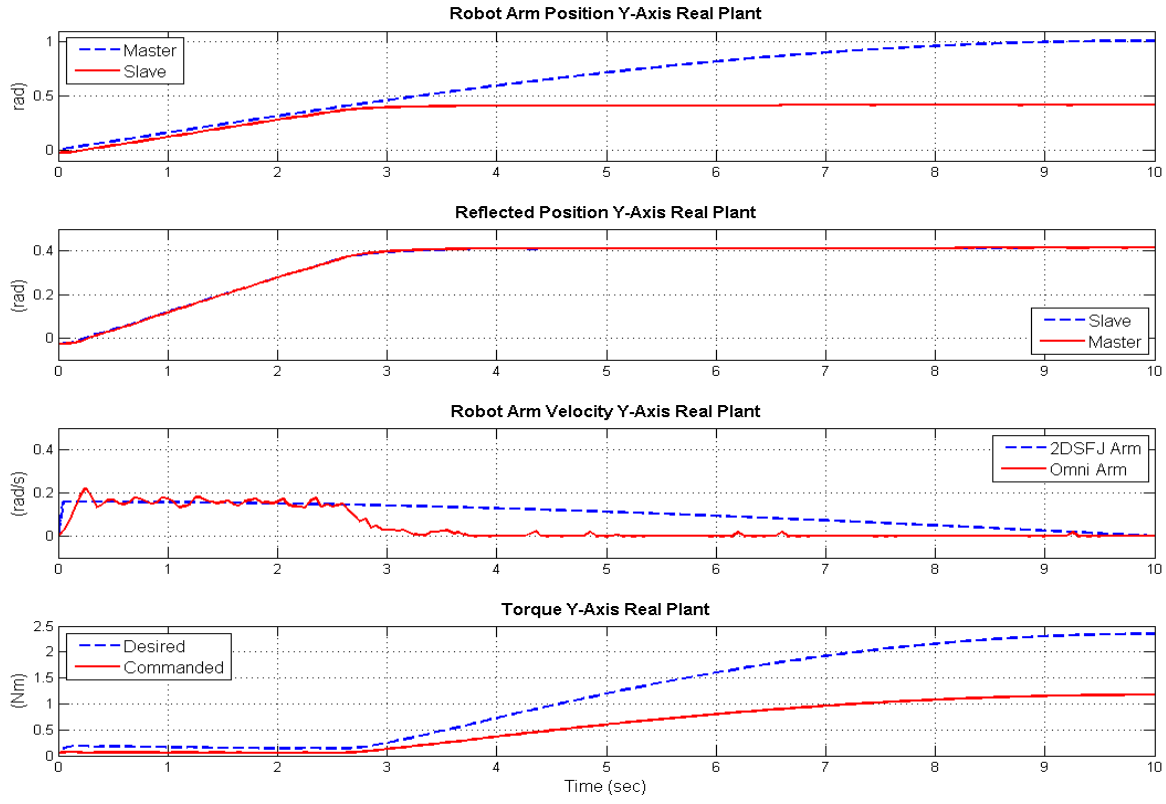


Figure 42: Scenario. 2 verification, Y-Axis real plant experiment plot

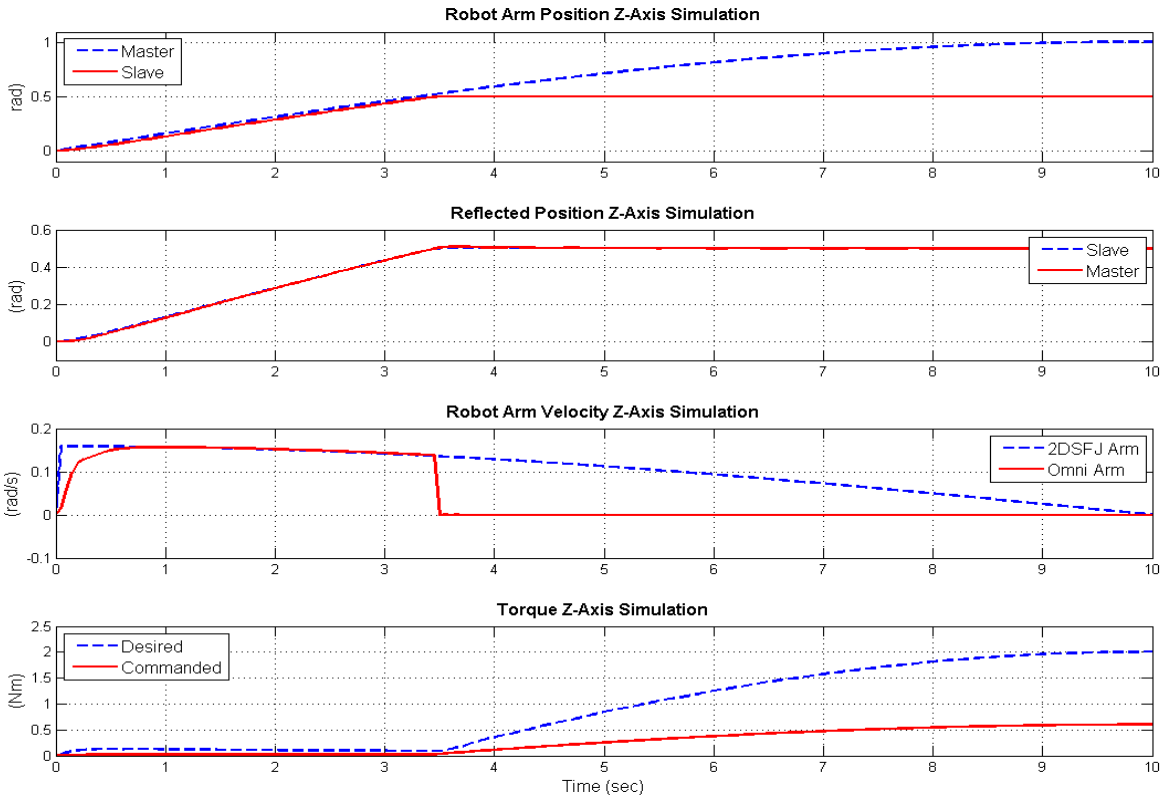


Figure 43: Scenario. 2 verification, Z-Axis simulation plot. Used model: `zAxisDemoModel`.

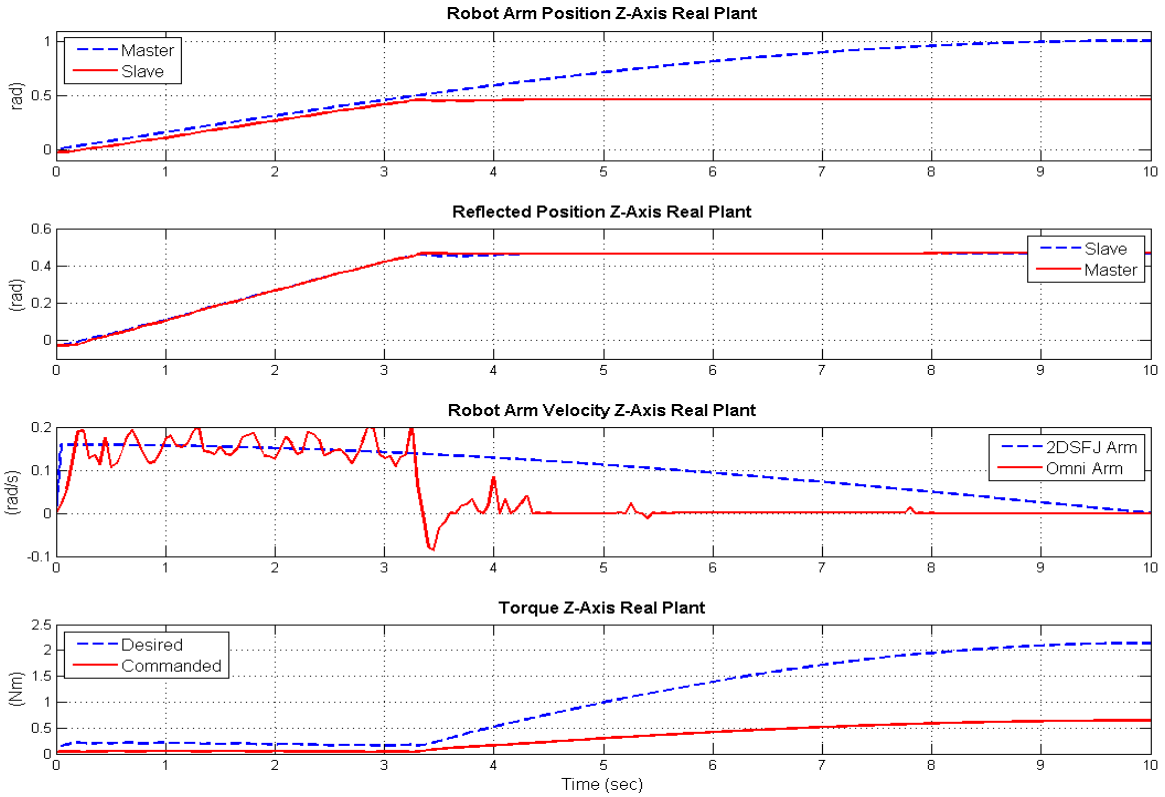


Figure 44: Scenario 2 verification, Z-Axis real plant experiment plot

As in 4.4.1, 3 regions can be identified and the following statements about the systems behaviour can be made:

Free space

- The position control gives good reference tracking results on both stages.
- The velocity signal form the slave is subject to disturbances, especially the Z-Axis.
- The position reflection is very accurate.

Transition

- The transition occurs smoothly. By stopping the slaves arm, the desired torque is build up slowly therefore also the environmental torque is build up slowly.

Contact with hard surface

- While in contact with a hard surface, the commanded torque is proportional to the desired torque as it should be.

5.4.2 Results: Manual mode

While operating the system in manual mode, the operator can make the following statements about the systems behaviour:

- The control scheme is intuitive.
- While in free space, the reflected position to the master causes that no resistant torque is felt by the operator.
- If the slave hits with a solid surface, the haptic experience leads to an intuitive stop on the master commands.

6 CONCLUSION

Within this work, a haptic experiment with two different robots was set up in place. One robot acts a master, the other as slave. Simulink is used in external mode to control the robots. Two operation scenarios were defined as master and slave may switch their roles. Each scenario contains one Simulink diagram for the simulation and one for the real plant experiment. A signal generator with GUI was implemented to have a good control over the artificial actuation signals used in almost all diagrams.

6.1 System Identification

The models, which represent the robots in the simulations, were found through a system identification process which uses captured experiment data to estimate and verify defined model structures. Grey box modelling was used which permits to define exact linear or nonlinear model structures with specific parameters. The model parameters were found using an estimation algorithm which starts with a linear model and reference data which do not treat the high nonlinearities of the real plants. However, more complex nonlinear models can be estimated if the initial guessed values were inside a certain region.

Demo models for each robot were created and verified. They gave a good representation of the real plants. Furthermore, the presented grey-box modeling process makes it easy to implement additional, more complex nonlinear model structures and to estimate their parameters if additional precision is needed for a specific actuation signal.

6.2 Controller design

The designed controller for both scenarios is based on force controlled haptic teleoperation, capable to control the robot in free space were no environmental force is present and while in contact with a hard surface. The transition from one situation to the other occurs without peaks or high oscillations. No force sensors are installed on the robots to achieve this task. The implemented controllers give a good performance inside the nonlinear environment. However, they are meant as performance comparison to more advanced controller designs.

6.3 Further steps

Based on this work, it is possible to define and estimate more complex models with the presented system identification process. These models can be used in the simulation diagrams to design more advanced controller setups, before testing and fine-tune them with the real plant experiment setups.

REMARK: As the Phantom Omni robot is put horizontally, and this is not the meant operating position, it is not suited for feedback operations very well. That means it slips over the table while applying a torque to the operator or environment. For serious experiment conditions, a support should be designed which holds the Omni in place.

Next, while experimenting with the Omni in horizontal mode for longer time periods, a sudden shut down of its amplifier was noticed (Could be related to an internal over limit protection). Dependant on the operation scenario this can lead to fast movements on the 2DSFJ robot (reflection). However, the 2DSFJ robot is protected by limiting switches but is recommended to use the saturation blocks to limit excessive loads.

7 DATE AND SIGNATURE

Calgary 27.September 2013

Franco Summermatter

franco.summermatter@gmail.com

8 REFERENCES

[1] D. Richert, C. J. Macnab and J. K. Pieper, "Adaptive Haptic Control for Telerobotics Transitioning Between Free, Soft and Hard Environments," *IEEE Trans. Syst. Man, Cybern. B, Syst., Humans., vol. 42, no. 3,* pp. 560-562, May 2012.

[2] SensAble, *PHANTOM Omni® User Guide,* Woburn, MA 01810, 2008.

[3] Quanser, *2-DOF Serial Flexible Joint Robot, Reference manual,* Revision 01.

[4] E. Wernholt and S. Gunnarsson, "Nonlinear Identification of a Physically Parameterized Robot Model," SYSID, Newcastle, Australia, 2006.

9 APPENDICES

9 APPENDICES

9.1 Appendix 1: Joint torsional stiffness verification measurement

Stage 1, weight attached at 150mm from joint axis.

Weight [g]	Force [N]	Torque [Nm]	deflection [rad]	Ks
50	0.491	0.074	0.022	3.391
100	0.981	0.147	0.035	4.204
150	1.472	0.221	0.052	4.245
200	1.962	0.294	0.072	4.116
250	2.453	0.368	0.094	3.901
			Average	**3.971**

Table 1: Stage 1 joint torsional stiffness measurements.

Stage 2, weight attached at 150mm from joint axis.

Weight [g]	Force [N]	Torque [Nm]	deflection [rad]	Ks
50	0.491	0.074	0.019	3.872
100	0.981	0.147	0.038	3.872
150	1.472	0.221	0.057	3.872
200	1.962	0.294	0.075	3.924
250	2.453	0.368	0.094	3.914
			Average	**3.891**

Table 2: Stage 2 joint torsional stiffness measurements.

9.2 Appendix 2: 2DSFJ Robot, physical parameters

Symbol	Description	Initial Value	Unit
I_{m1}	Stage1: Motor current	±0.94 (max)	A
K_{tg1}	Stage1: Drive torque constant (harmonic drive output)	8.925	Nm/A
K_{s1}	Stage1: Joint torsional stiffness	3.9264	Nm/rad
θ_{11}	Stage1: Drive Shaft absolute angular position	± π/2 (max)	rad
$\dot{\theta_{11}}$	Stage1: Drive shaft absolute angular velocity		rad/s
θ_{12}	Stage1: Flexible joint absolute angular position	± π/2 (max)	rad
$\dot{\theta_{12}}$	Stage1: Flexible joint absolute angular velocity		rad/s
J_{11}	Stage1: Actuated transition, moment of inertia	0.011	$Kg \cdot m^2$
B_{11}	Stage1: Actuated transition, viscous damping coeff.	4.500	$Nm \cdot s/rad$
J_{12}	Stage1: Flexible joint, moment of inertia (with stage2)	0.230	$Kg \cdot m^2$
B_{12}	Stage1: Flexible joint, viscous damping coeff.	0.070	$Nm \cdot s/rad$
I_{m2}	Stage2: Motor current	±1.2 (max)	A
K_{tg2}	Stage2: Drive torque constant (harmonic drive output)	0.87	Nm/A
K_{s2}	Stage1: Joint torsional stiffness	3.9264	Nm/rad
θ_{21}	Stage2: Drive Shaft absolute angular position	± π/2 (max)	rad
$\dot{\theta_{21}}$	Stage2: Drive shaft absolute angular velocity		rad/s
θ_{22}	Stage2: Flexible joint absolute angular position	± π/2 (max)	rad
$\dot{\theta_{22}}$	Stage2: Flexible joint absolute angular velocity		rad/s
J_{21}	Stage2: Actuated transition, moment of inertia	0.009	$Kg \cdot m^2$
B_{21}	Stage2: Actuated transition, viscous damping coeff.	0.5	$Nm \cdot s/rad$
J_{22}	Stage2: Flexible joint, moment of inertia	0.011	$Kg \cdot m^2$
B_{22}	Stage2: Flexible joint, viscous damping coeff.	0.028	$Nm \cdot s/rad$

Table 3: 2DSFJ physical parameter

9.3 Appendix 3: 2DSFJ, iddata object actuation signal settings:

Stage 1

st1DemoData:

```
2DSFJ robot: C_2DSFJ_Data_Aqu_SM, stage1 data aquisition
simulation duration: 60 s
frequency sweep:     1.5 - 0.1 Hz over 46 s
amplitude sweep:     0.08*sine(2*pi*0.1) A
saturation:          none
```

st1DemoData2:

```
2DSFJ robot: C_2DSFJ_Data_Aqu_SM, stage1 data aquisition
simulation duration:      60 s
frequency sweep:     1.2 - 0.1 Hz over 30 s
amplitude sweep:     0.08*sine(2*pi*0.2) A
saturation:          none
```

Stage 2

st2DemoData:

```
2DSFJ robot: C_2DSFJ_Data_Aqu_SM, stage2 data aquisition
simulation duration:      60 s
frequency sweep:     1.5 - 0.1 Hz over 38 s
amplitude sweep:     0.2*sine(2*pi*0.1) A
saturation:          none
```

st2DemoData2:

```
2DSFJ robot: C_2DSFJ_Data_Aqu_SM, stage2 data aquisition
simulation duration:      60 s
frequency sweep:     1 - 0.1 Hz over 30 s
amplitude sweep:     0.15*sine(2*pi*0.15) A
saturation:          none
```

9.4 Appendix 4: 2DSFJ, Stage 1 model parameter for demo estimation comparison

modSt1Init

Parameters:		value		
p1	Ktg	8.925	(fix)	in [0, 60]
p2	Ks	3.9264	(fix)	in [0, 60]
p3	J1	0.0111	(fix)	in [0, 60]
p4	B1	4.5	(fix)	in [0, 60]
p5	J2	0.2304	(fix)	in [0, 60]
p6	B2	0.0704	(fix)	in [0, 60]

modSt2DampOnly

Parameters:		value	standard deviation		
p1	Ktg	8.925	0	(fix)	in [0, 60]
p2	Ks	3.9264	0	(fix)	in [0, 60]
p3	J1	0.0111	0	(fix)	in [0, 60]
p4	B1	1.99687	0.0232304	(est)	in [0, 60]
p5	J2	0.2304	0	(fix)	in [0, 60]
p6	B2	5.02247e-008	0.0166659	(est)	in [0, 60]

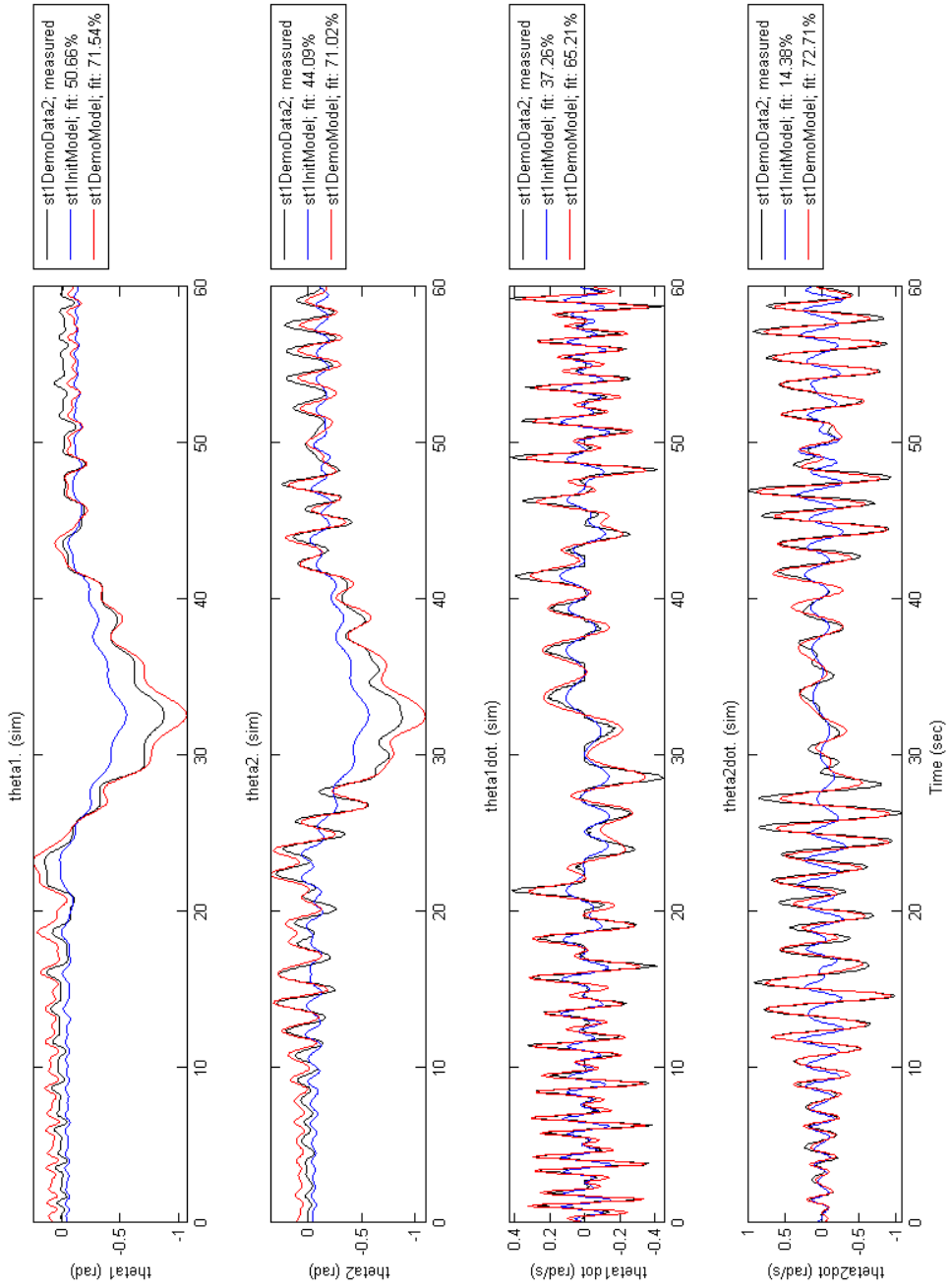
modSt1All

Parameters:		value	standard deviation		
p1	Ktg	58.6024	189.482	(est)	in [0, 60]
p2	Ks	3.59016	11.6172	(est)	in [0, 60]
p3	J1	0.357069	1.15409	(est)	in [0, 60]
p4	B1	13.3387	43.1117	(est)	in [0, 60]
p5	J2	0.2812	0.910114	(est)	in [0, 60]
p6	B2	0.0390078	0.127554	(est)	in [0, 60]

9.5 Appendix 5: 2DSFJ *st1DemoModel*, parameters and verification plot

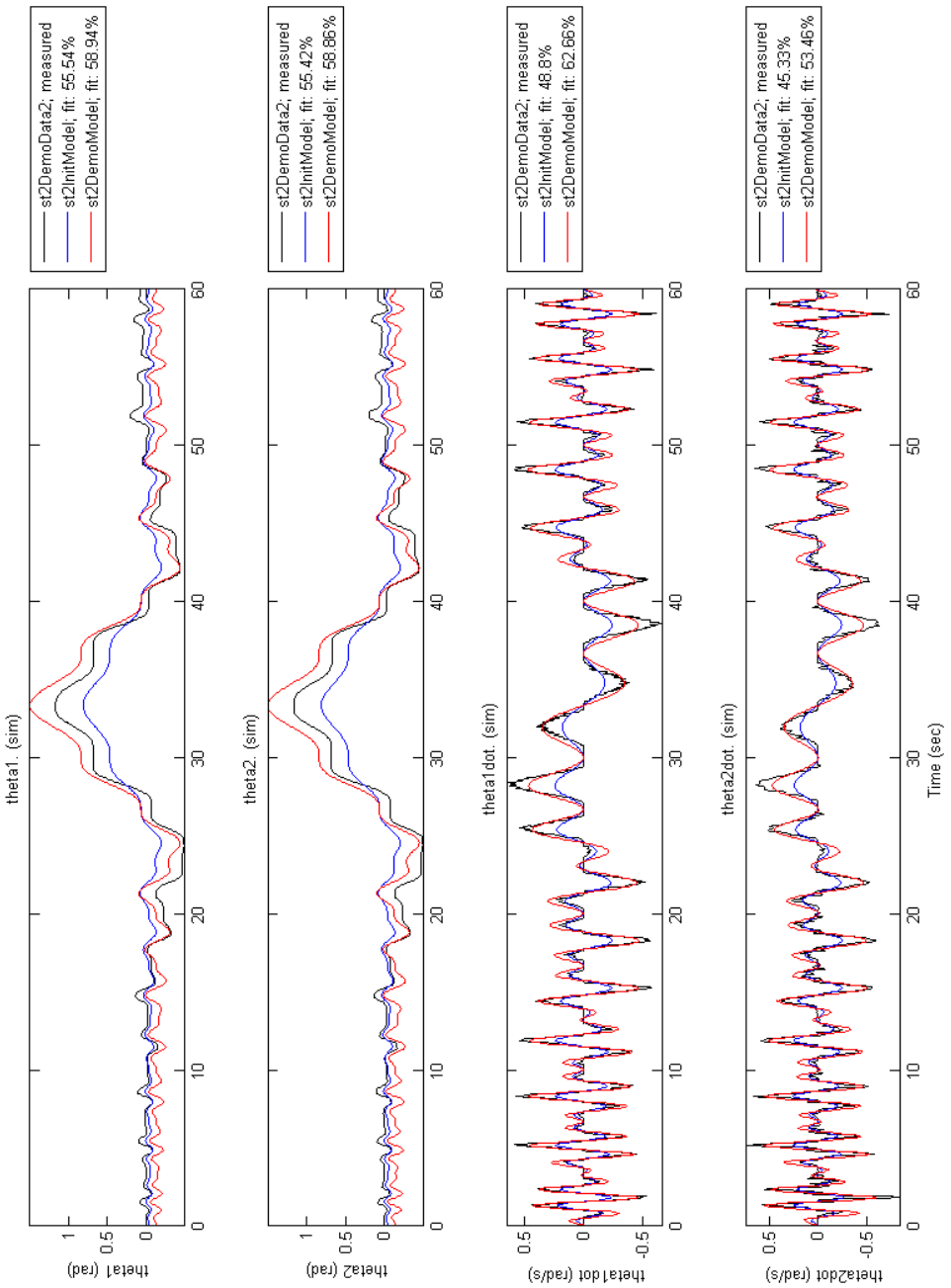
```
initial states: Estimate

    Parameters:      value       standard deviation
p1  Ktg          31.7126          1.26547    (est) in [0, 60]
p2  Ks           3.9264           0          (fix) in [0, 60]
p3  J1           0.211465         0.0140016  (est) in [0, 60]
p4  B1           7.1033           0.280763   (est) in [0, 60]
p5  J2           0.302584         0.00103729 (est) in [0, 60]
p6  B2           0.0380436        0.00701162 (est) in [0, 60]
```



9.6 Appendix 6: 2DSFJ *st2DemoModel*, parameters and verification plot

```
initial states: Zero
    Parameters:       value        standard deviation
p1    Ktg          4.53598       2.16337    (est) in [0, 60]
p2    Ks            3.9264             0    (fix) in [0, 60]
p3    J1          0.013634     0.0234865    (est) in [0, 60]
p4    B1           1.43588      0.680867    (est) in [0, 60]
p5    J2         0.0234559    0.00087938    (est) in [0, 60]
p6    B2     1.28651e-005    0.00752049    (est) in [0, 60]
```



9.7 Appendix 7: Phantom Omni iddata object actuation signal settings

Y-axis

yAxisDemoData

```
Phantom OMNI: C_Omni_Data_Acq_SM, y-Axis data aquisition
simulation duration:60 s
frequency sweep:     1.5 - 0.01 Hz over 60 s
amplitude sweep:     0.4*sine(2*pi*0.1) rad
saturation:          none
detrended
```

yAxisDemoData2

```
Phantom OMNI: C_Omni_Data_Acq_SM, y-Axis data aquisition
simulation duration:60 s
frequency sweep:     0.1 - 1 Hz over 60 s
amplitude sweep:     0.4*sine(2*pi*0.2) rad
saturation:          none
detrended
```

Z-axis

zAxisDemoData

```
Phantom OMNI: C_Omni_Data_Acq_SM, y-Axis data aquisition
simulation duration:60 s
frequency sweep:     2 - 0.01 Hz over 55 s
amplitude sweep:     0.4*sine(2*pi*0.1) rad
saturation:          none
detrended
```

zAxisDemoData2

```
Phantom OMNI: C_Omni_Data_Acq_SM, y-Axis data aquisition
simulation duration:60 s
frequency sweep:     0.1 - 2 Hz over 55 s
amplitude sweep:     0.4*sine(2*pi*0.2) rad
saturation:          none
detrended
```

9.8 Appendix 8: Omni, Y-Axis model parameter for demo estimation comparison

Linear model

greyOMNI parameters:

Parameters:		value	standard deviation	
p1	Jm	0.00247149	0.0015654	(est) in [0, 10]
p2	Ja	0.00067359	0.000598149	(est) in [0, 10]
p3	Bg	0.0184404	0.016579	(est) in [0, 10]
p4	Bm	0.0460532	0.00151621	(est) in [0, 10]
p5	Kg	0.002312	0.00208609	(est) in [0, 10]

Nonlinear model

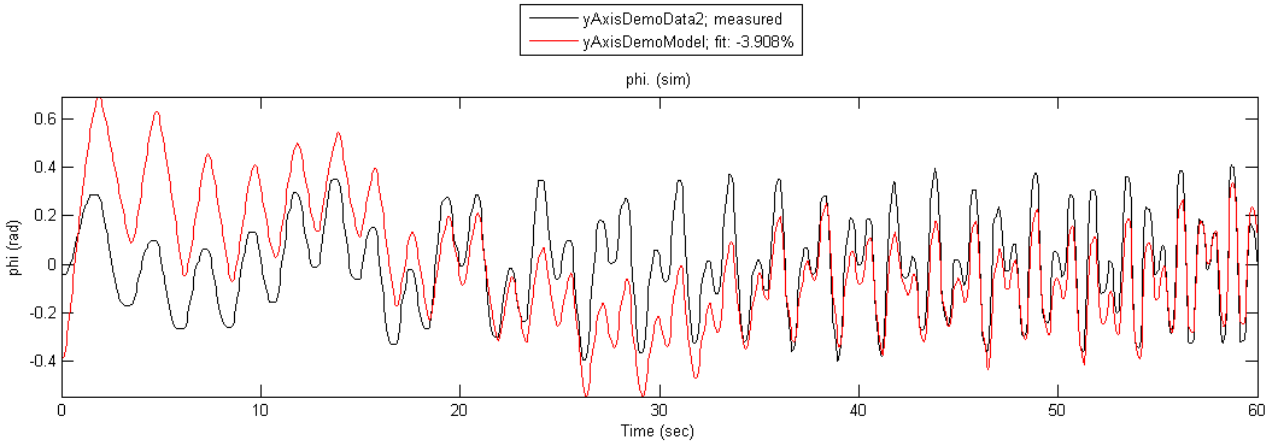
greyOMNInl parameters:

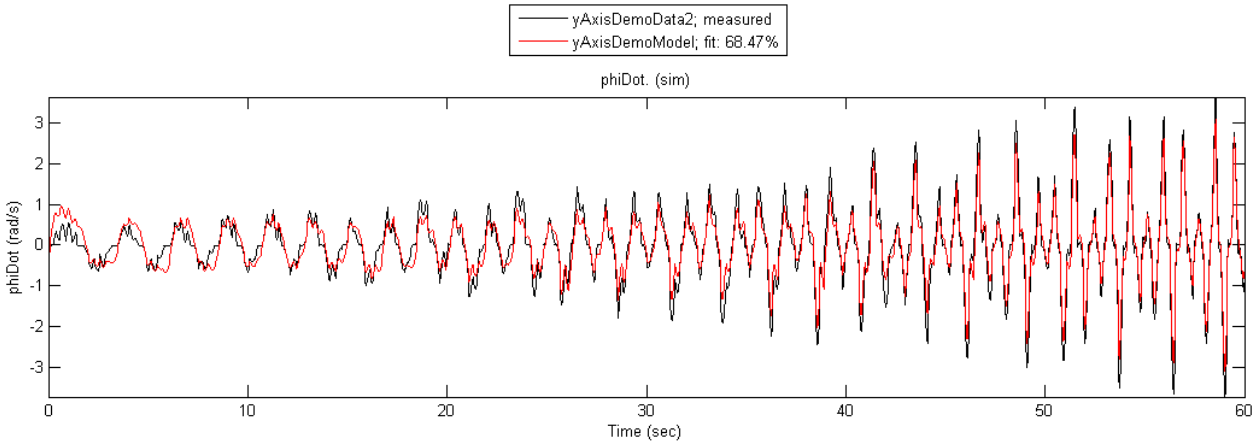
Parameters:		value	standard deviation	
p1	Jm	0.000504815	8.61438e-005	(est) in [0, 10]
p2	Ja	0.000398497	8.83154e-005	(est) in [0, 10]
p3	Da	0.00296561	0.000645844	(est) in [0, 10]
p4	Ka	0.0285382	0.00696336	(est) in [0, 10]
p5	Fc	0.762169	24.4584	(est) in [0, 10]
p6	Fv	0.0243613	0.541876	(est) in [0, 10]
p7	Fcs	6.36503	32.5752	(est) in [0, 10]
p8	alpha	4.10178	0.128355	(est) in [0, 10]
p9	beta	0.0224495	0.114824	(est) in [0, 10]

9.9 Appendix 9: Omni *yAxisDemoModel*, parameters and verification plot

```
initial states: Estimate

Parameters:       value          standard dev
  p1    Jm        0.00126386    9.83032e-005    (est) in [0, 10]
  p2    Ja      2.10079e-005     0.000122922    (est) in [0, 10]
  p3    Da       0.000308567      0.00181747    (est) in [0, 10]
  p4    Ka       0.000155434     0.000915847    (est) in [0, 10]
  p5    Fc          0.920368          164.45    (est) in [0, 10]
  p6    Fv         0.0156475          1.11491    (est) in [0, 10]
  p7    Fcs          7.21912          466.978    (est) in [0, 10]
  p8    alpha        1.55978       0.0776333    (est) in [0, 10]
  p9    beta      0.00933871        0.603933    (est) in [0, 10]
```



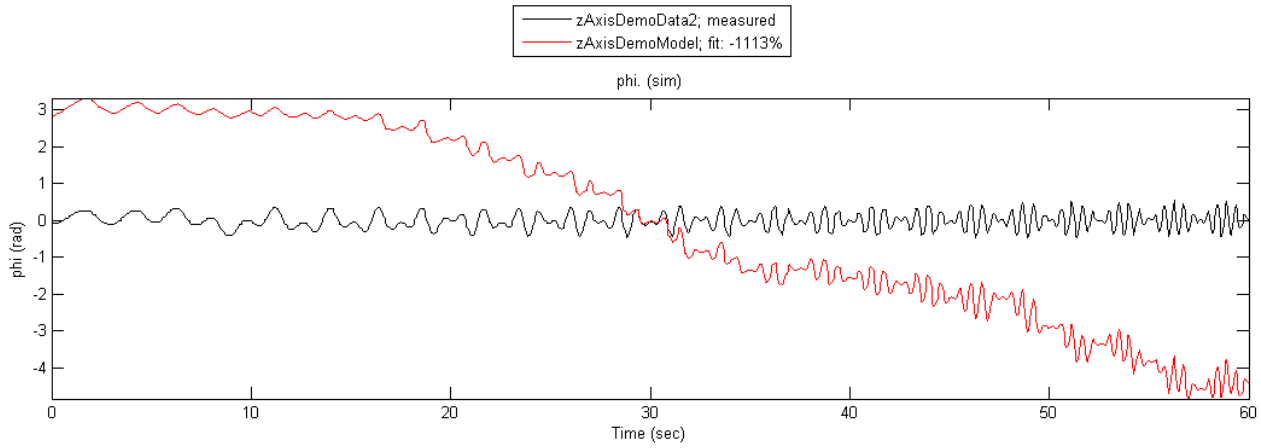


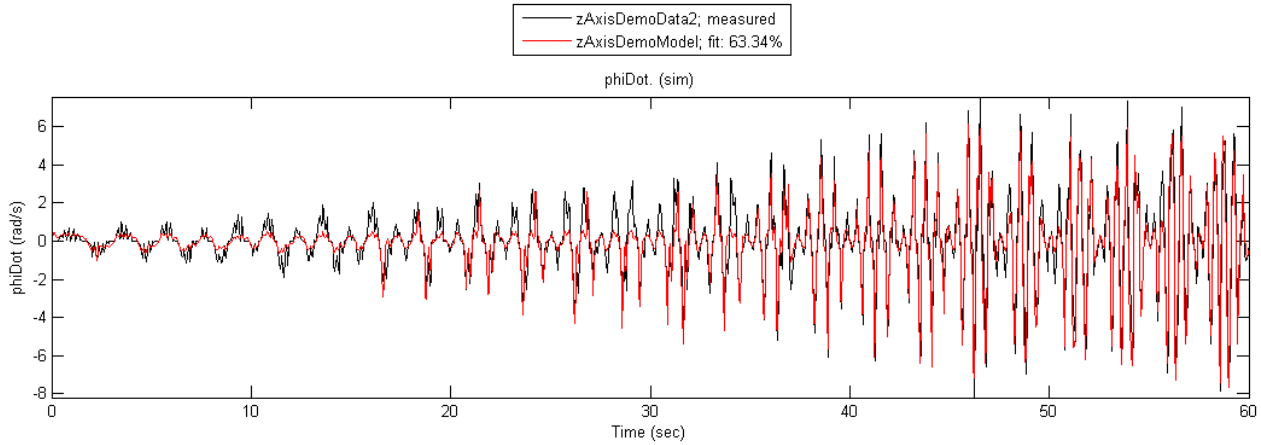
9.10 Appendix 10: Omni *zAxisDemoModel*, parameters and verification plot

```
initial states: Estimate

Parameters:       value          standard dev

  p1    Jm       0.000307573    2.26662e-005    (est) in [0, 10]
  p2    Ja       0.000205652    7.74974e-006    (est) in [0, 10]
  p3    Da        0.00418913     0.000257804    (est) in [0, 10]
  p4    Ka         0.0554702      0.00224884    (est) in [0, 10]
  p5    Fc           0.59724         4.55998    (est) in [0, 10]
  p6    Fv        0.00256818       0.0919813    (est) in [0, 10]
  p7    Fcs          6.96003         1.27281    (est) in [0, 10]
  p8    alpha        2.00401       0.0434375    (est) in [0, 10]
  p9    beta       0.0206379      0.00380606    (est) in [0, 10]
```





9.11 Appendix 11: Scenario 1, controller parameter

Parameter	Description	Value
ST1_lambda	Ratio between desired torque and motor velocity in free space	0.5
ST1_KP_1	PI-controller, proportional gain	0
ST1_Ki	PI-controller, integrator gain	20
ST1_Kp_2	Position error, Proportional gain	0.25
ST1_Kd	Velocity term, proportional gain	0.1

Table 4: Scenario 1 Stage 1 controller, parameters.

Parameter	Description	Value
ST2_lambda	Ratio between desired torque and motor velocity in free space	0.5
ST2_KP_1	PI-controller, proportional gain	0
ST2_Ki	PI-controller, integrator gain	20
ST2_Kp_2	Position error, Proportional gain	0.25
ST2_Kd	Velocity term, proportional gain	0.1

Table 5: Scenario 1 Stage 2 controller parameters.

9.12 Appendix 12: Scenario 2, controller parameter

Reflection Controller

Parameter	Description	Value
ST1_KPreflect	Proportional gain	5
ST1_KIreflect	Integral gain	5
ST1_KDreflect	Derivative gain	0.1

Table 6: Scenario 2 Stage 1 reflection controller parameters.

Parameter	Description	Value
ST2_KPreflect	Proportional gain	5
ST2_KIreflect	Integral gain	5
ST2_KDreflect	Derivative gain	0

Table 7: Scenario 2 Stage 2 reflection controller parameters.

Axes Controller

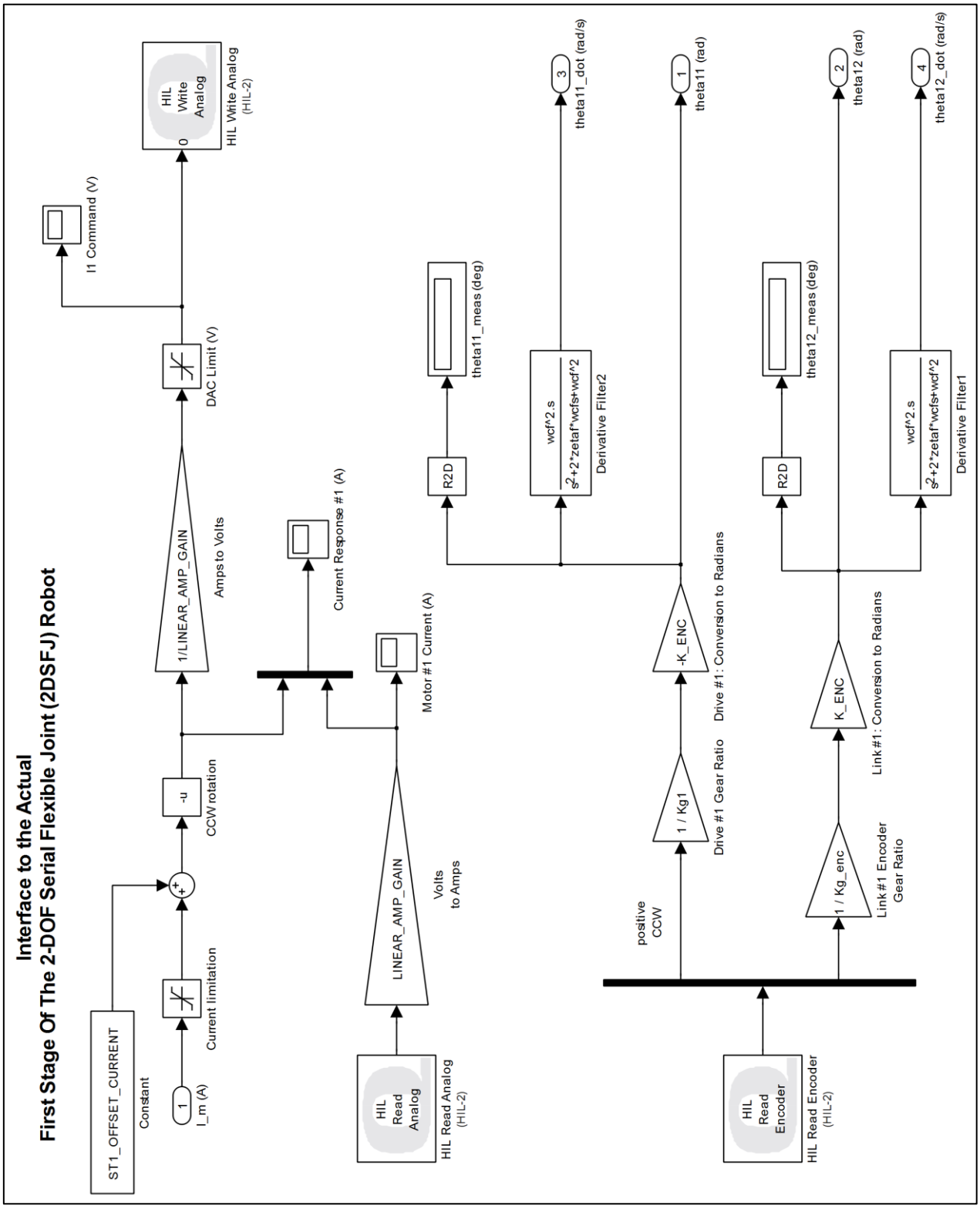
Parameter	Description	Value
Y_Kp1	Controller gain	0.1
Y_Lambda	Torque gain	0.5

Table 8: Scenario 2 Y-Axis controller parameters.

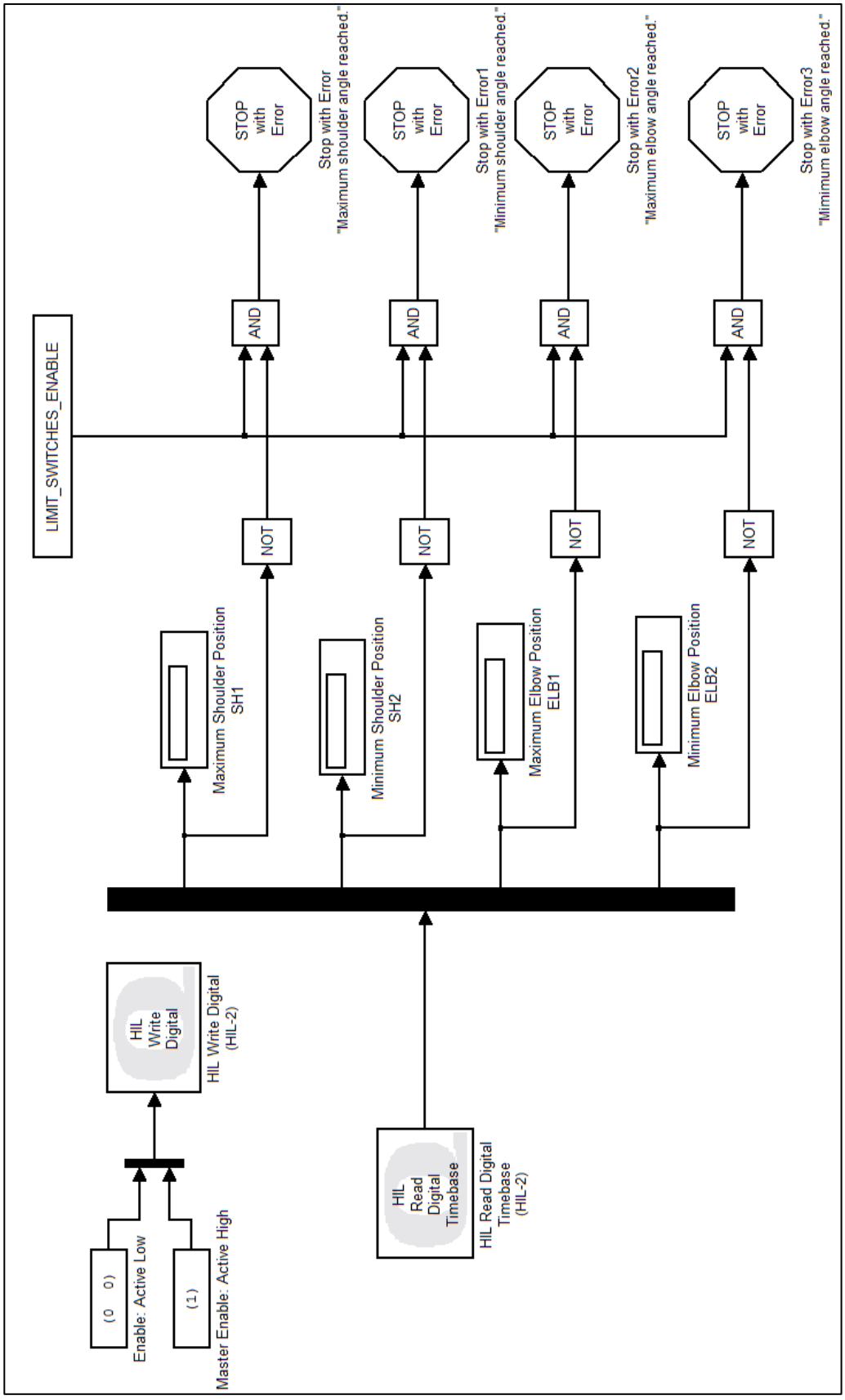
Parameter	Description	Value
Z_Kp1	Controller gain	0.05
Z_Lambda	Torque gain	0.3

Table 9: Scenario 2 Z-Axis controller parameters.

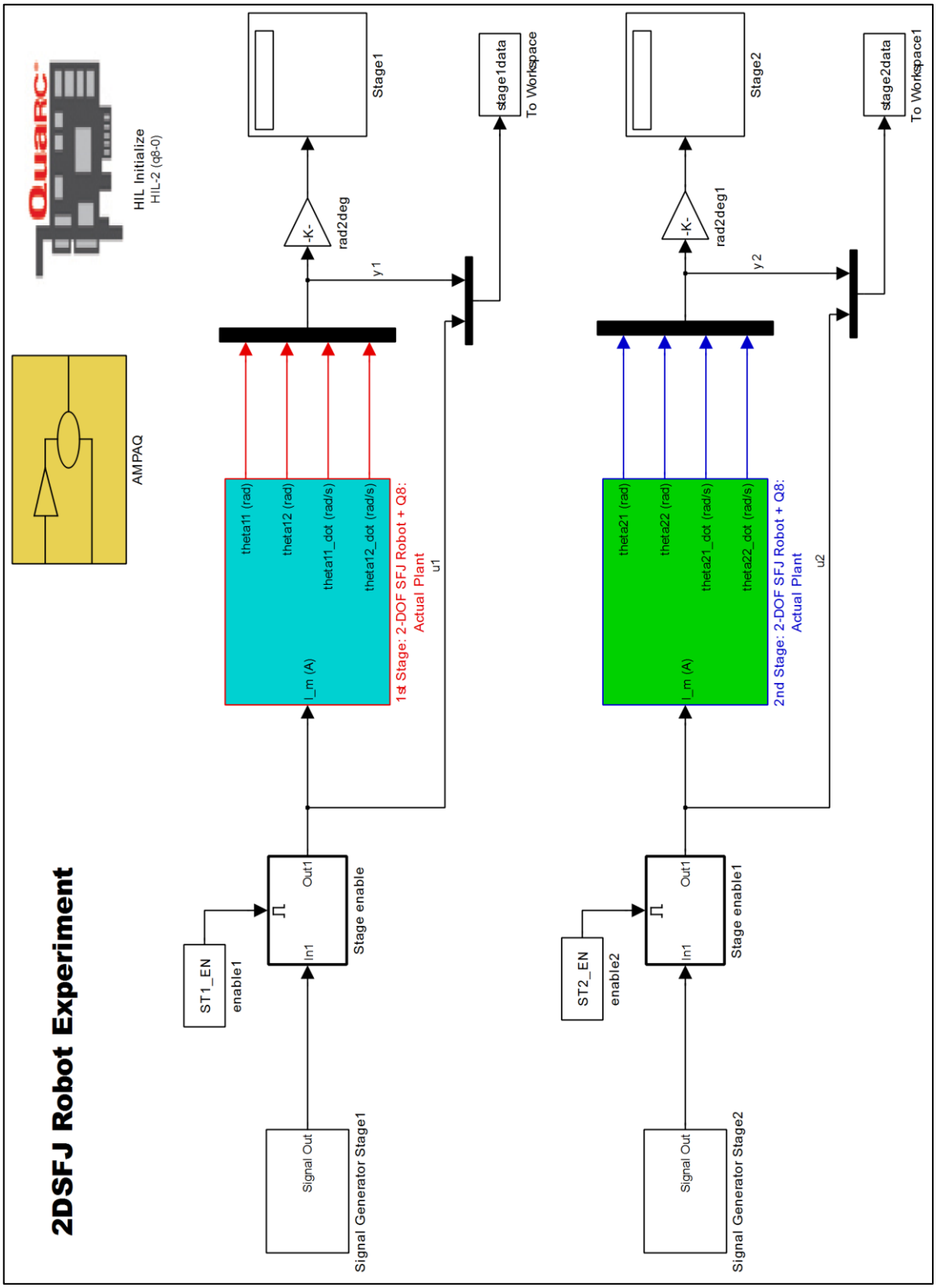
9.13 Appendix 13: Simulink diagram, 2DSFJ robot, Stage block



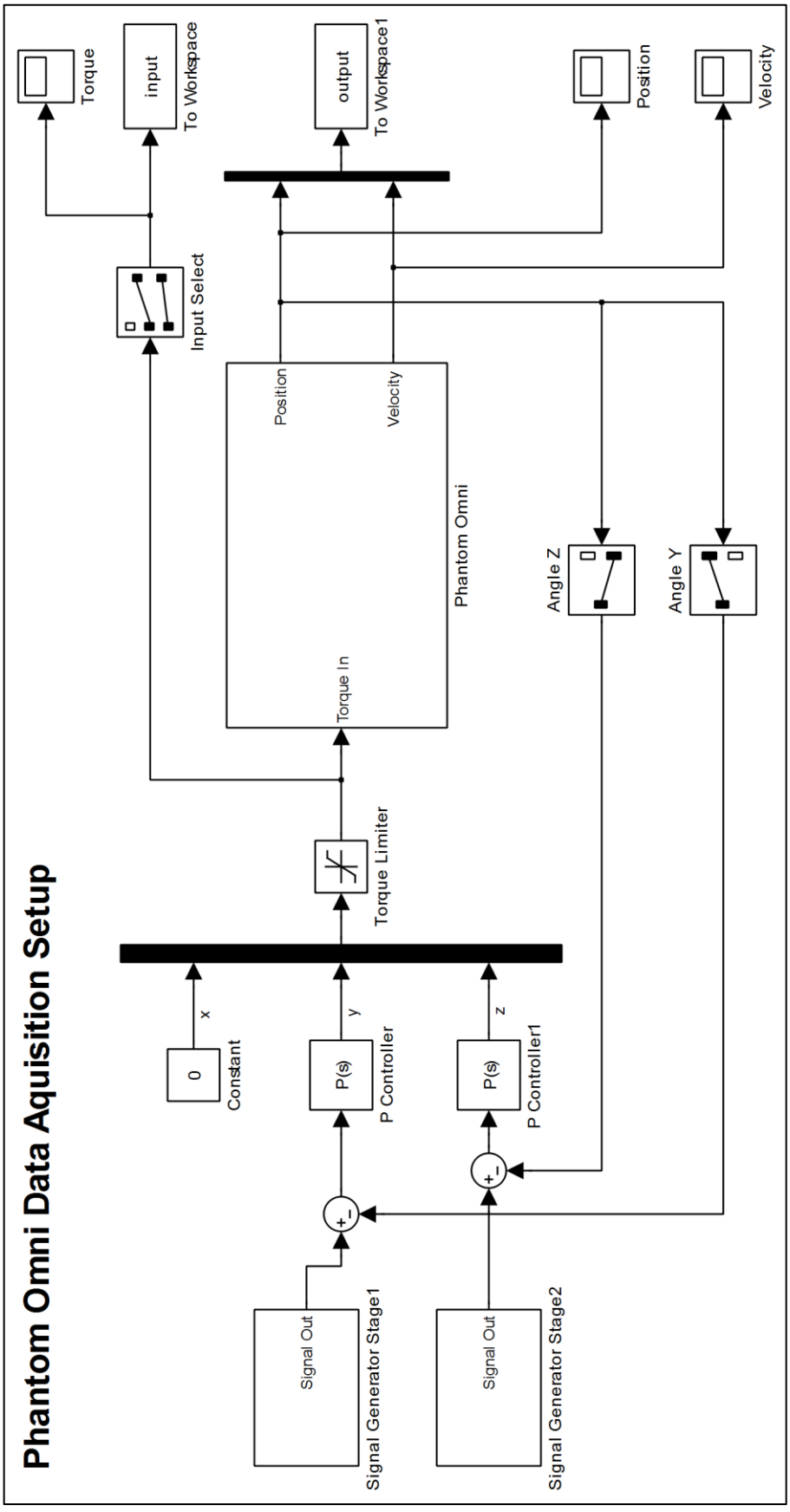
9.14 Appendix 14: Simulink diagram, AMPAQ block



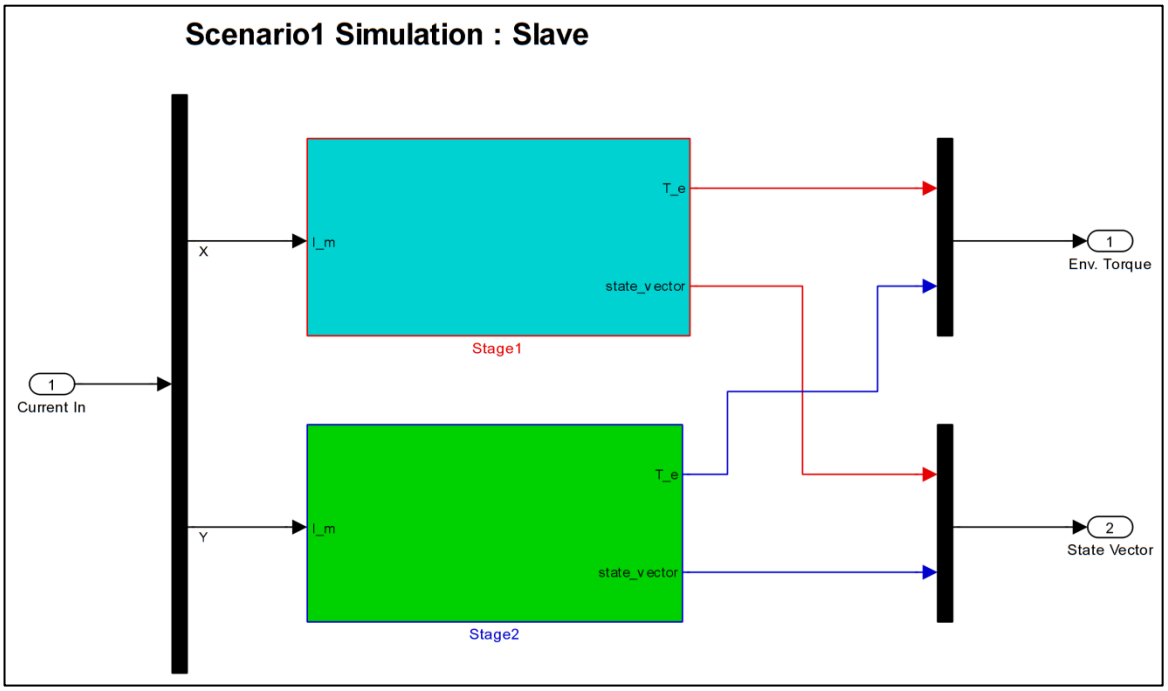
9.15 Appendix 15: Simulink diagram, *C_2DSFJ_Data_Acq_SM*



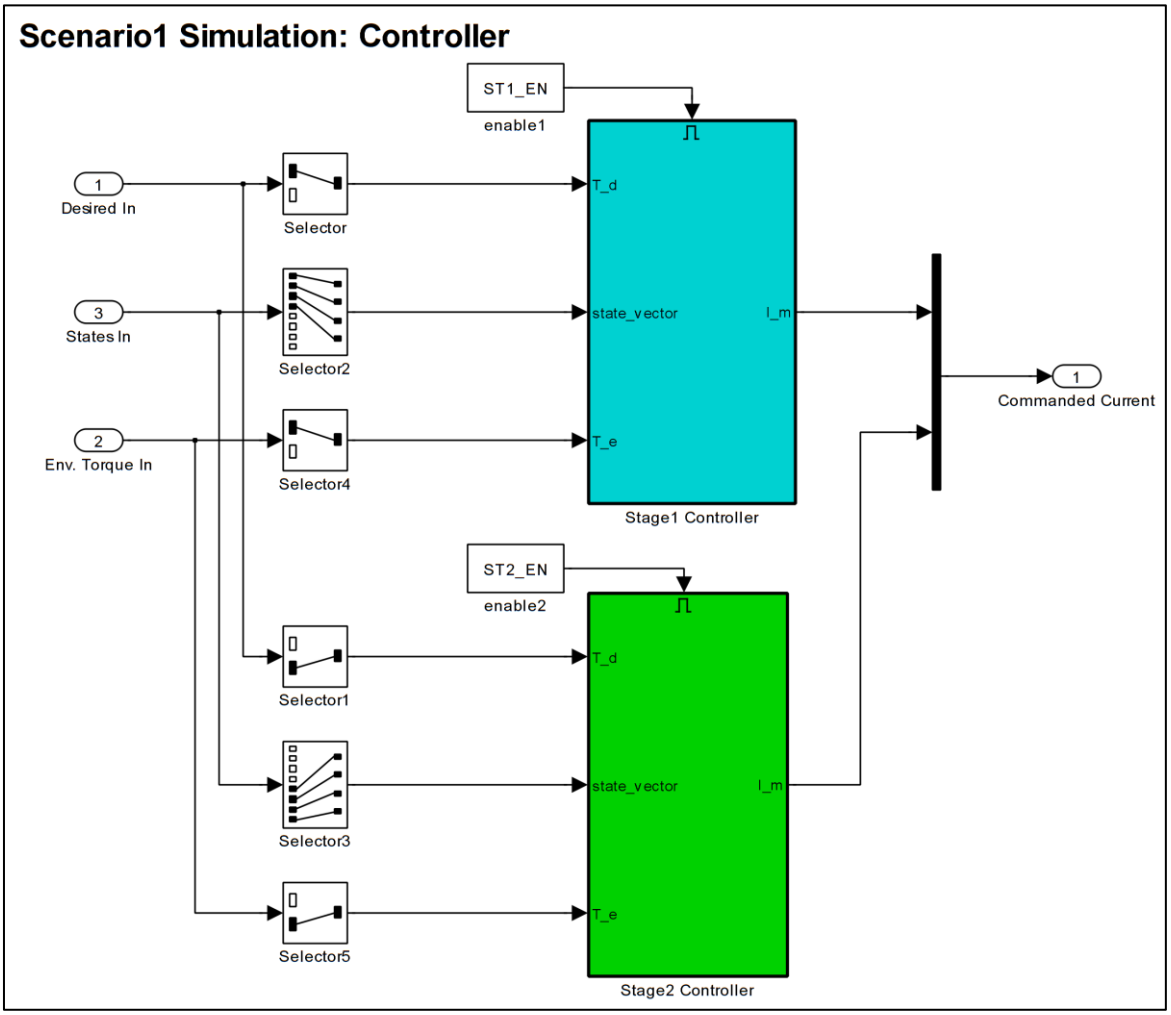
9.16 Appendix 16: Simulink diagram, *C_Omni_Data_Acq_SM*



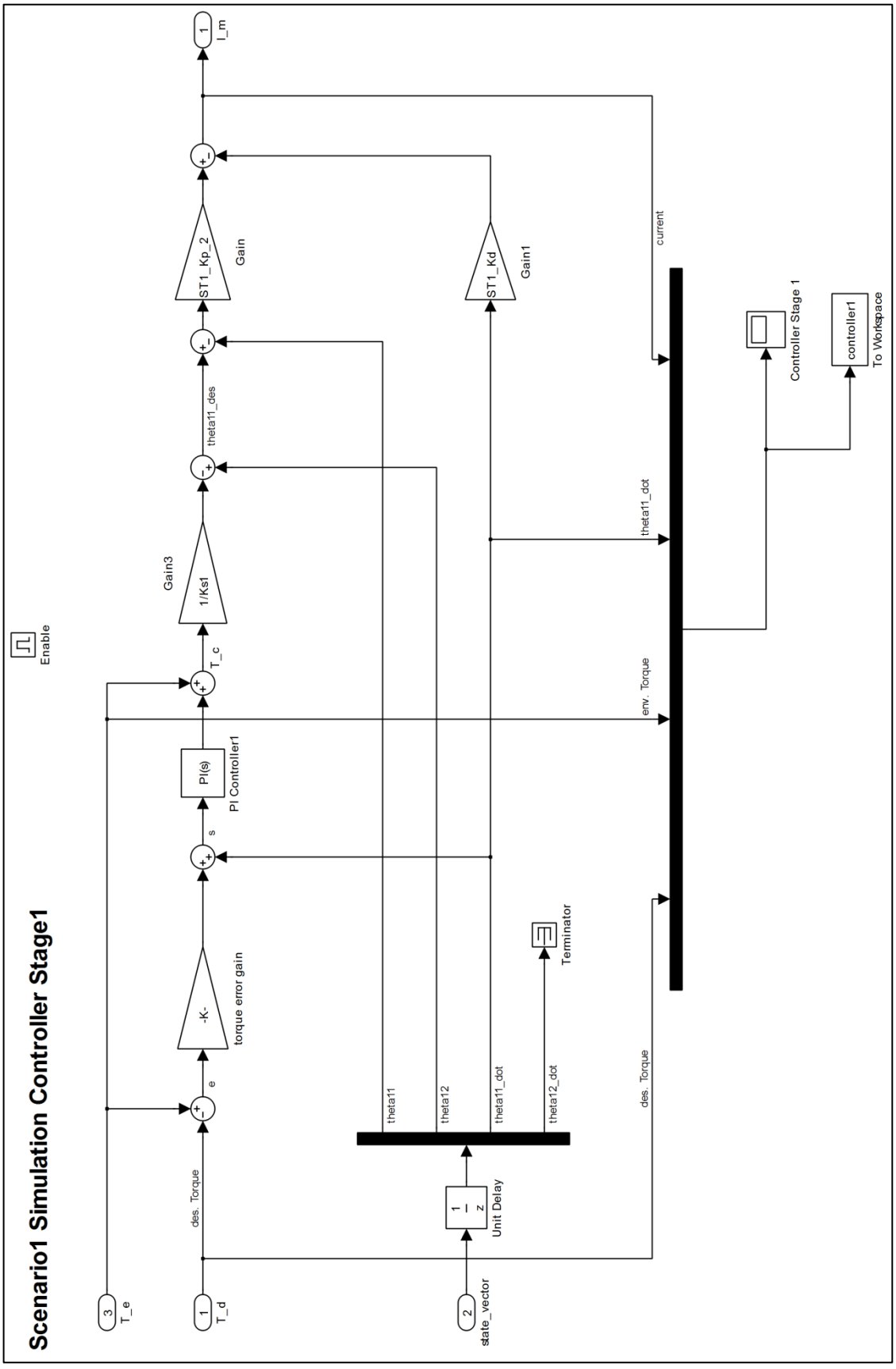
9.17 Appendix 17: Simulink diagram, *Scenario 1 Slave top level*



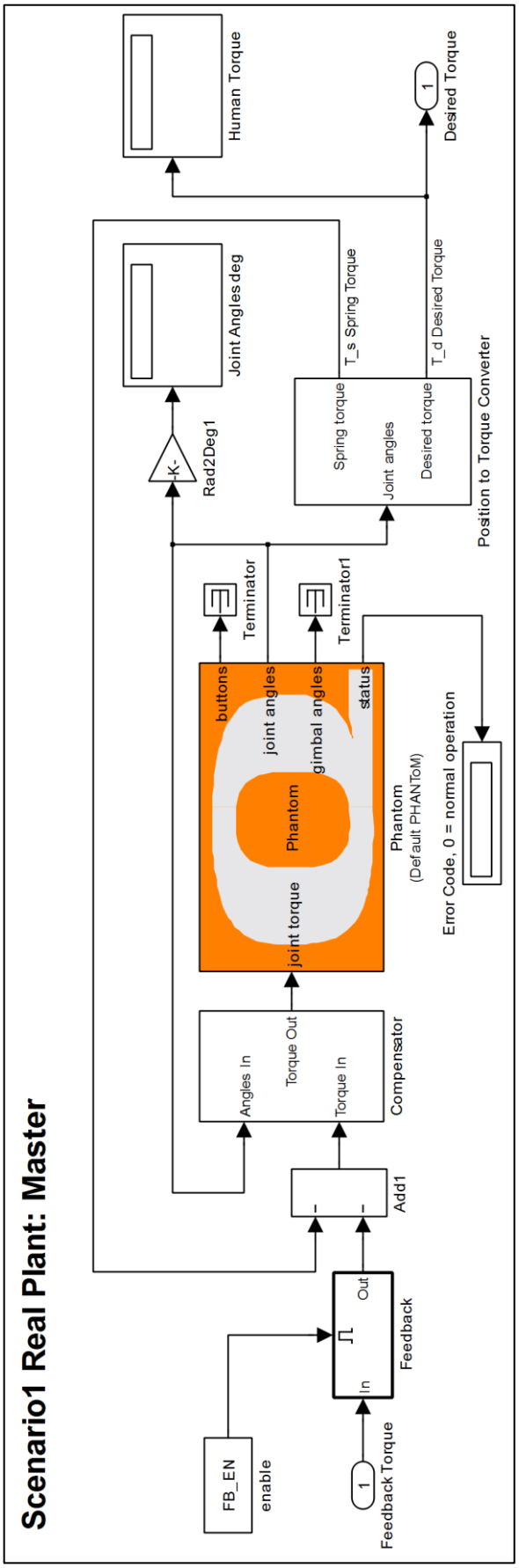
9.18 Appendix 18: Simulink diagram, *Scenario 1 Controller top level*



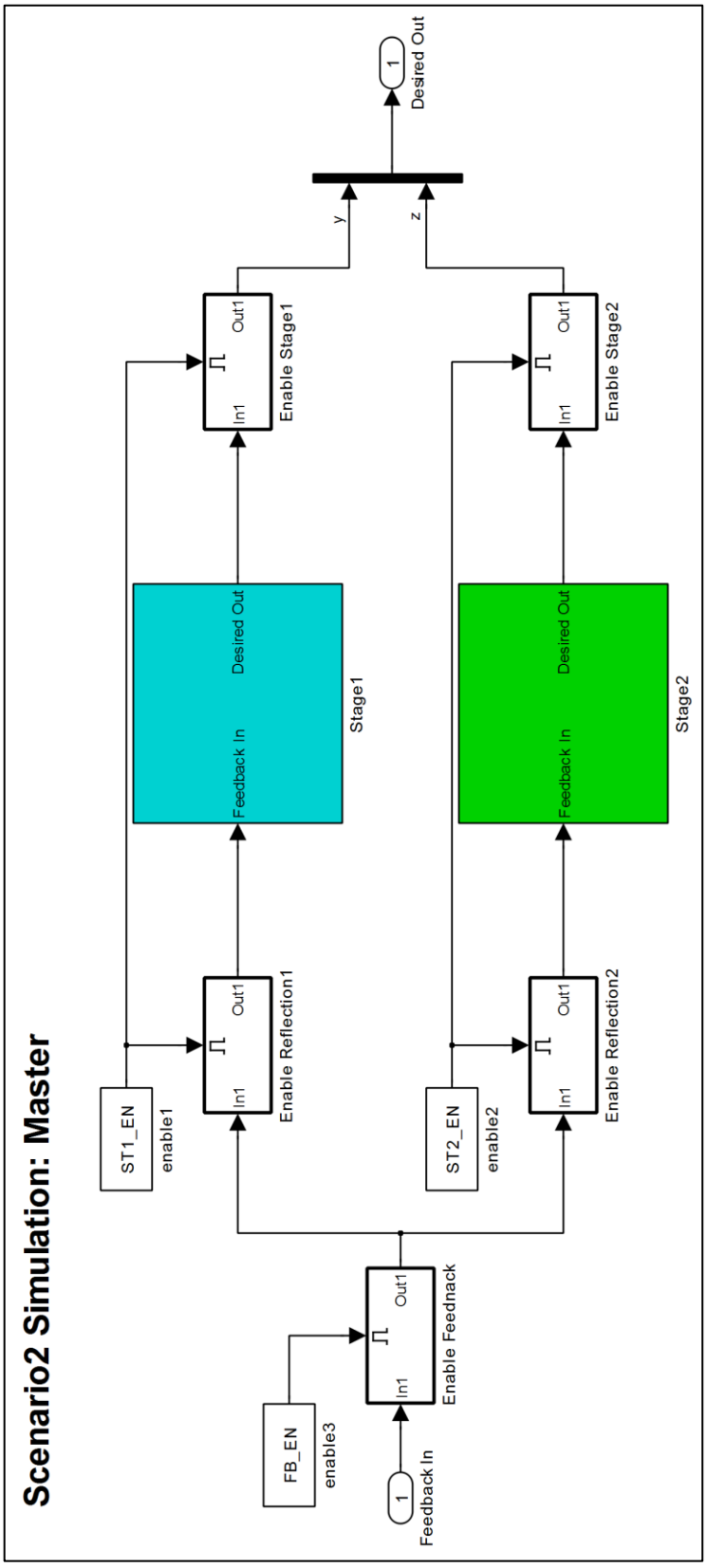
9.19 Appendix 19: Simulink diagram, *Scenario 1 Controller Stage1*



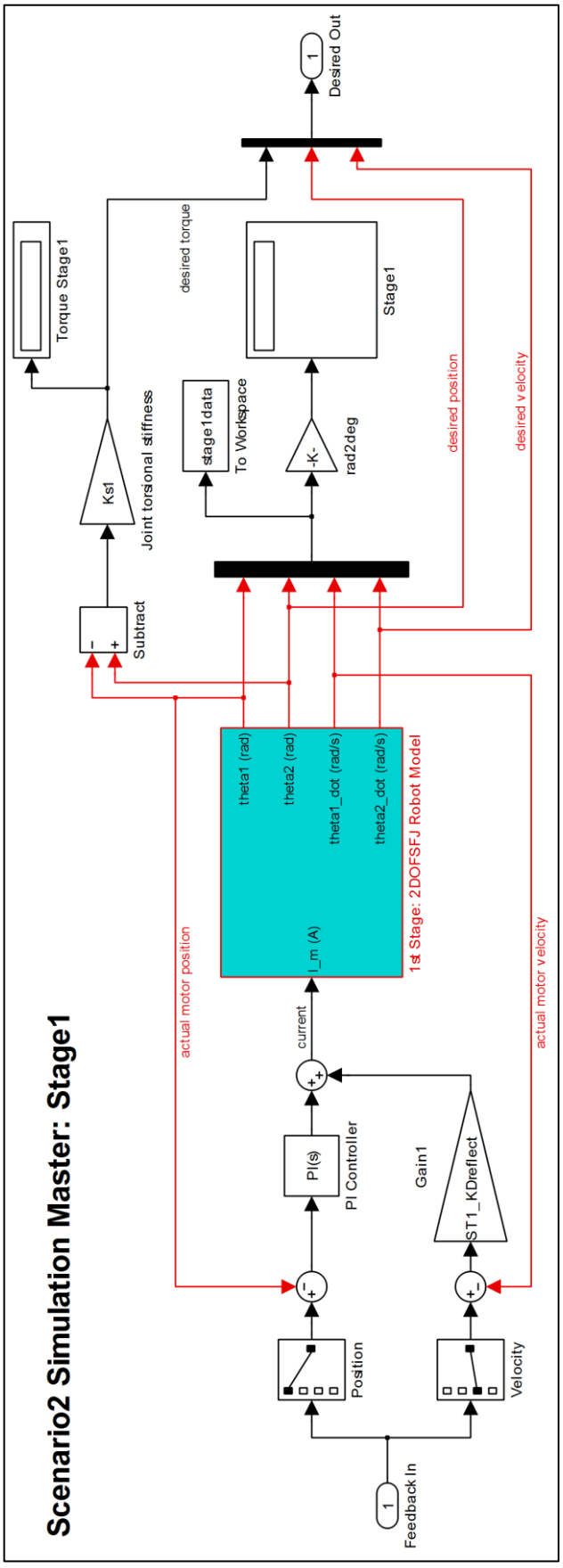
9.20 Appendix 20: Simulink diagram, *Scenario 1 Real Plant Master*



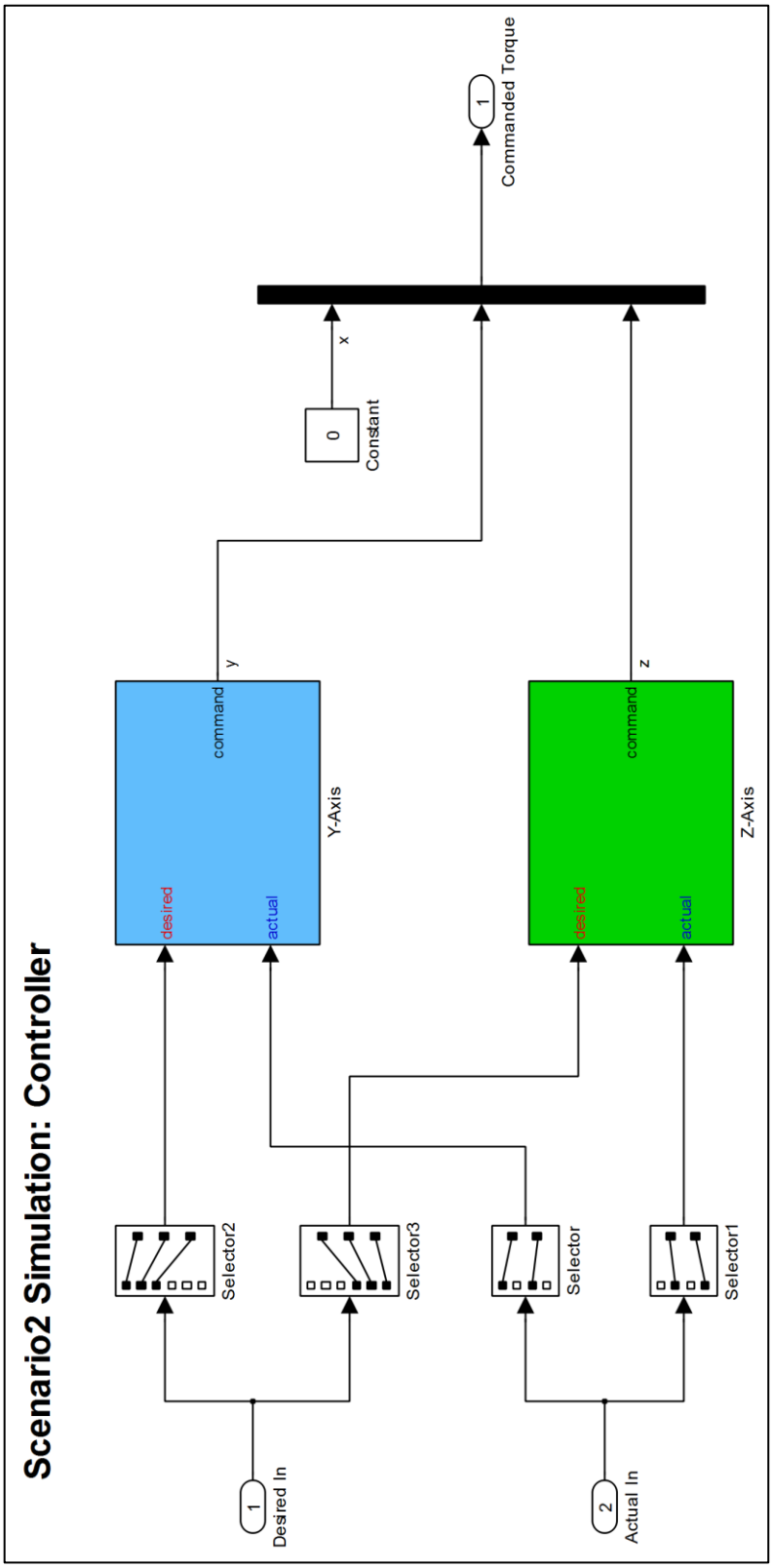
9.21 Appendix 21: Simulink diagram, *Scenario 2 Master top level*



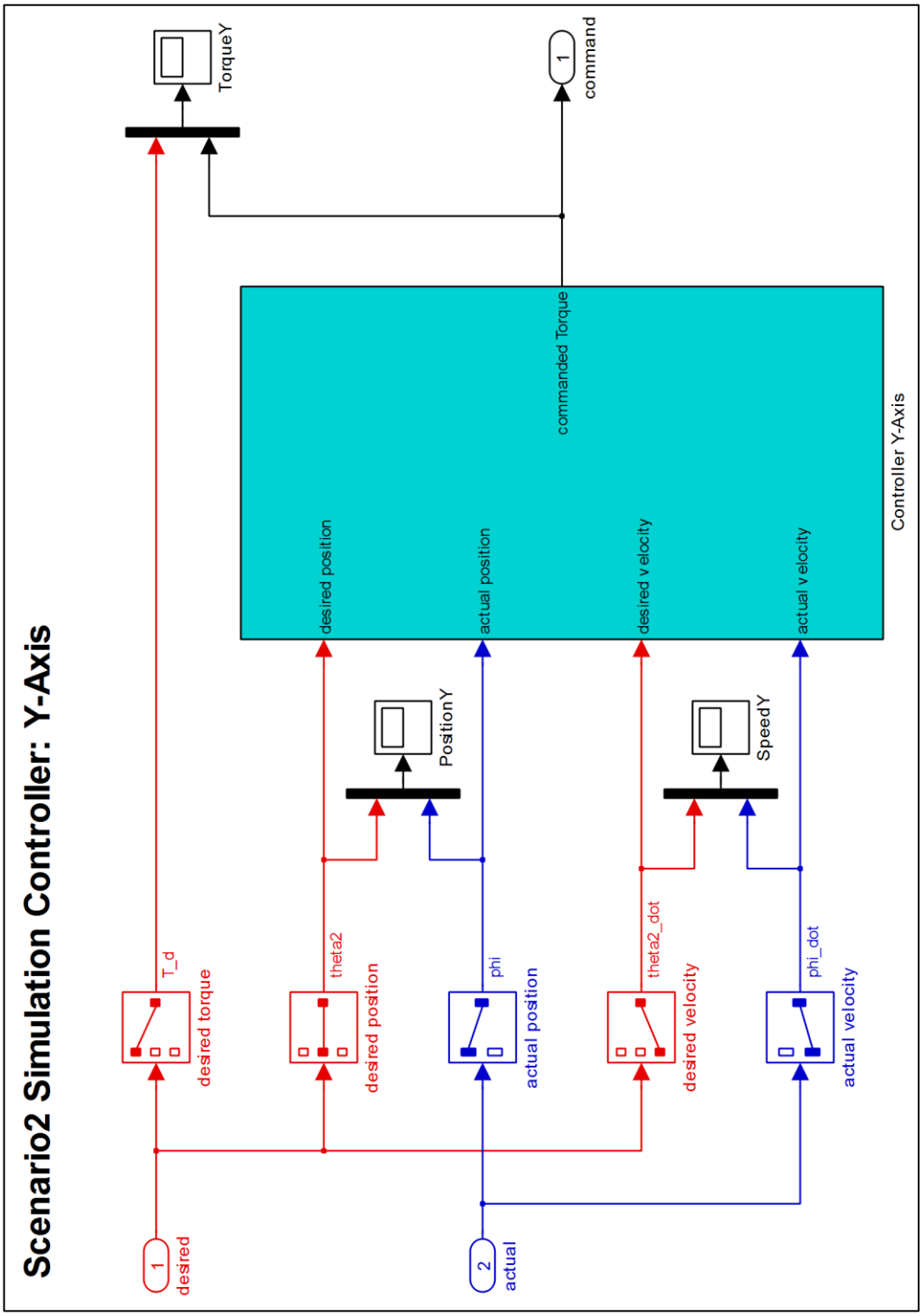
9.22 Appendix 22: Simulink diagram, *Scenario 2 Master one Stage 1*



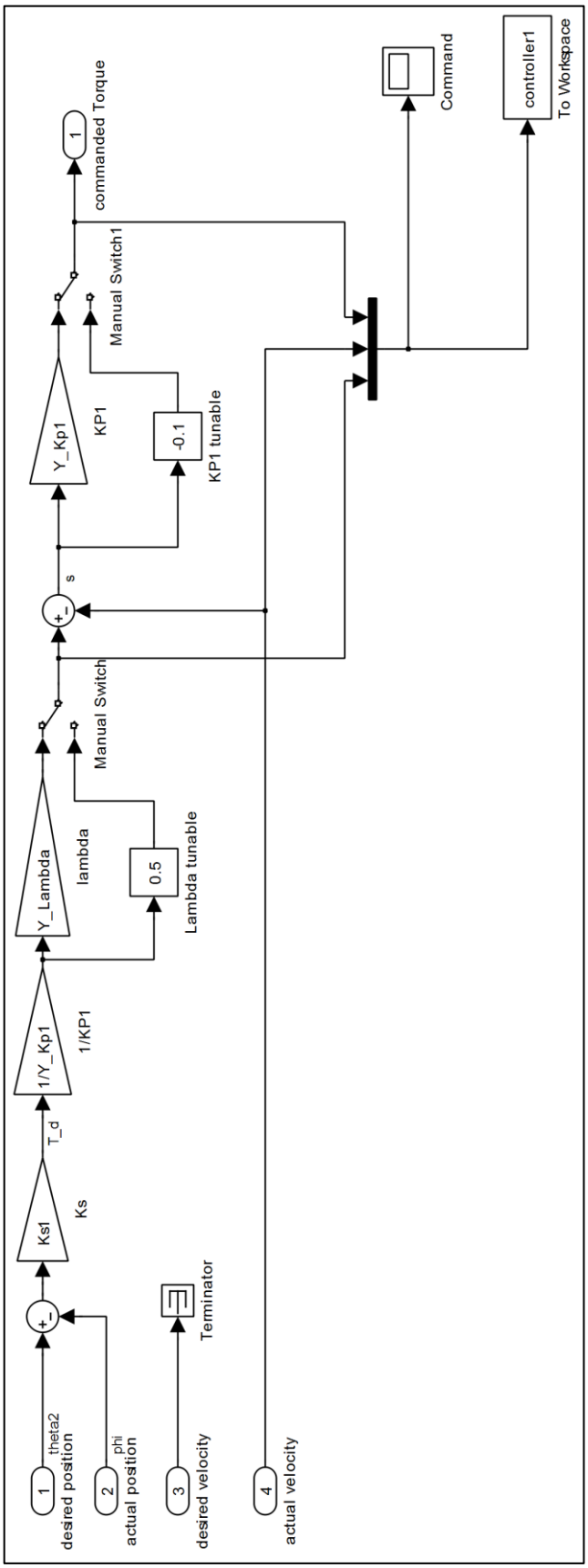
9.23 Appendix 23: Simulink diagram, *Scenario 2 Controller top level*



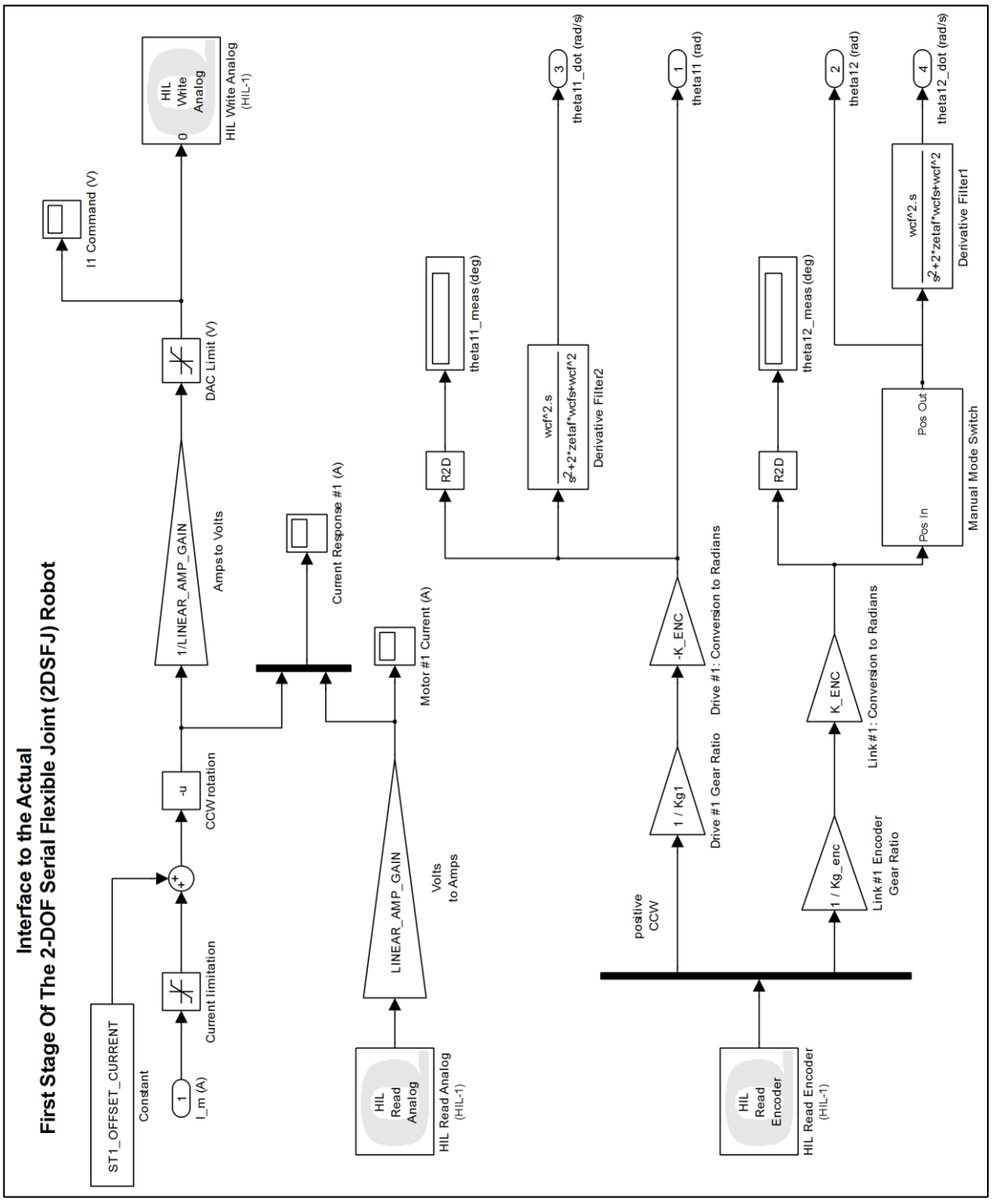
9.24 Appendix 24: Simulink diagram, *Scenario 2 Controller Y-Axis*



9.25 Appendix 25: Simulink diagram, *Scenario 2 Controller block Y-Axis*



9.26 Appendix 26: Simulink diagram, *Scenario 2 Real Plant Master Stage1*



9.27 Appendix 27: MATLAB code, *constants*

```
1.   %% Constants Definitions
2.   SIM_STEP = 0.002;           % Simulation: Fixed step size, default 0.002s
3.
4.  % these following names and units are used for data sets AND the grey box
5.  % models. In order to verify the models, input and output names and units
6.  % must be the same in data and models objects.
7.
8.  INPUT_NAME_OMNI = {'torque'};
9.  INPUT_UNIT_OMNI = {'Nm'};
10. OUTPUT_NAME_OMNI = {'phi' 'phiDot'};
11. OUTPUT_UNIT_OMNI = {'rad' 'rad/s'};
12.
13. INPUT_NAME_2DSFJ = {'current'};
14. INPUT_UNIT_2DSFJ = {'A'};
15. OUTPUT_NAME_2DSFJ = {'theta1' 'theta2' 'theta1dot' 'theta2dot'};
16. OUTPUT_UNIT_2DSFJ = {'rad' 'rad' 'rad/s' 'rad/s'};
17.
18. disp('Constants loaded to workspace...');
```

9.28 Appendix 28: MATLAB code, *robot_2DSFJ_setup*

```
%% 2DSFJ Robot Parameters

V_MAX = 28;                 % AMPAQ maximum output voltage (V)
LINEAR_AMP_GAIN = 0.5;      % [A/V], AMPAQ
LIMIT_SWITCHES_ENABLE = 1;  % all four limit switches are enabled

% offset current to prevent drifting in one side
ST1_OFFSET_CURRENT = -0.0021584;
ST2_OFFSET_CURRENT = -0.002;

% spring attachment distance from joint centerline to hole 1 (m)
d1 = 86.36e-3;
% spring attachment radius: distance from joint axis to hole 1 (m)
r1 = 97.60e-3;
% spring attachment distance from joint centerline to hole 2 (m)
d2 = 73.66e-3;
% spring attachment radius: distance from joint axis to hole 2 (m)
r2 = 85.56e-3;
% spring attachment distance from joint centerline to hole 3 (m)
d3 = 60.96e-3;
% spring attachment radius: distance from joint axis to hole 3 (m)
r3 = 74.91e-3;
% converting factor: from lb/in to N/m
LBPIN_TO_NPM = 175.127;
% spring rate for the heaviest string (E0240-031-1500-S) (N/m)
Ra1 = 3.5 * LBPIN_TO_NPM; % = 612.94
% spring rate for the medium string (E0240-029-1500-S) (N/m)
Ra2 = 2.42 * LBPIN_TO_NPM; % = 423.81
% spring rate for the lightest string (E0240-026-1500-S) (N/m)
Ra3 = 1.33 * LBPIN_TO_NPM; % = 232.92

% Harmonic drive #1 gear ratio
Kg1 = 100;

% First Flexible Joint Torsional Stiffness (N.m/rad)
% default spring configuration: hole 1 - hole 1 mounting, lightest spring
Ks1_th = 2 * Ra3 * r1 * d1; % theoretical value: 3.9264
Ks1 = Ks1_th;   % 3.97; % measured average value

% fictitious Spring constant
Kfs1 = 1*Ks1_th;

% drive #1 maximum continuous current (A)
I1_MAX = 0.2;   % 0.94 max;

% Harmonic drive #2 gear ratio
Kg2 = 80;

% Second Flexible Joint Torsional Stiffness (N.m/rad)
% default spring configuration: hole 1 - hole 1 mounting, lightest spring
Ks2_th = 2 * Ra3 * r1 * d1; % theoretical value: 3.9264
Ks2 = Ks2_th;%3.891; % measured average value

% drive #2 maximum continuous current (A)
I2_MAX = 0.3;   % 1.2 max;

% Motor and Link (#1 and #2) Encoder Resolution (before gear ratios)
K_ENC = 2 * pi / ( 4 * 1024 ); % = 0.0015 rad/count
% gear ratio to the flexible joint encoder (#1 and #2)
Kg_enc = 6.23;

% Drive #1 (after gearbox) Encoder Sensitivity (rad/count)
KE_D1 = K_ENC / Kg1; % = 1.5340e-005
% Drive #2 (after gearbox) Encoder Sensitivity (rad/count)
```

```
65. KE_D2 = K_ENC / Kg2; % = 1.9175e-005
66. % Link (#1 and #2) (after gearbox) Encoder Sensitivity (rad/count)
67. KE_L = K_ENC / Kg_enc; % = 2.3968e-004
68.
69. % Specifications of a second-order low-pass filter
70. wcf = 2 * pi * 20;      % Velocity filter cutting frequency, stage1
71. wcf2 = 2 * pi * 20;     % Velocity filter cutting frequency, stage2
72. zetaf = 0.9;            % filter damping ratio
73.
74. disp('2DSFJ parameter loaded to workspace...');
```

9.29 Appendix 29: MATLAB code, *phantomOmni_setup*

```
%% Phantom Omni Parameters

%% Compensator Block Parameters:
x0 = -0.23;          % x axis reference angle [rad]
Kp_x0 = 4;           % PI Controller parameter to lock the x axis
Ki_x0 = 0.5;

y_MAX = 1.8;         % y axis max angle [rad]
Ks_y = 0.08;         % compensation spring const. [Nm/rad]

Tc_z = -0.01;        % z axis compensation torque [Nm]

%% Angle Offset Correction:
% neutral positions, [deg]
y0 = 50;     % relative to ground plane
z0 = 65;     % relative to vertical axis

offsetY = y0/180*pi;
offsetZ = z0/180*pi;

%% Velocity Filter
% Specifications of a second-order low-pass filter
wcf_o = 2 * pi * 20;     % Velocity filter cutting frequency, Phantom Omni
zetaf_o = 0.9;           % filter damping ratio

disp('Phantom Omni parameter loaded to workspace...');
```

9.30 Appendix 30: MATLAB code, *SignalGeneratorGUI*

GUI Opening function, load default parameters from workspace and update the GUI.

```
36. % --- Executes just before SignalGeneratorGUI is made visible.
37. function SignalGeneratorGUI_OpeningFcn(hObject, eventdata, handles,...
38.     varargin)
39. % This function has no output args, see OutputFcn.
40. % hObject    handle to figure
41. % eventdata  reserved - to be defined in a future version of MATLAB
42. % handles    structure with handles and user data (see GUIDATA)
43. % varargin   command line arguments to SignalGeneratorGUI (see VARARGIN)
44.
45. % Choose default command line output for SignalGeneratorGUI
46. handles.output = hObject;
47.
48. % set default parameters form workspace
49. set(handles.edit_simDuration, 'String',...
50.     num2str(evalin('base', 'SIM_DURATION')));
51. set(handles.edit_sigDelay,         'String',         num2str(evalin('base',
    'SIM_DELAY')));
52.
53. for i = 1:2
54.     j = num2str(i);
55.     set(eval(['handles.cb_st' j '_enable']), 'Value',...
56.         evalin('base', ['ST' j '_EN']));
57.     set(eval(['handles.cb_st' j '_ampSweep']), 'Value',...
58.         evalin('base', ['ST' j '_A_SWEEP_EN']));
59.     set(eval(['handles.cb_st' j '_freqSweep']), 'Value',...
60.         evalin('base', ['ST' j '_F_SWEEP_EN']));
61.
62.     set(eval(['handles.edit_st' j '_amp']), 'String',...
63.         num2str(evalin('base', ['ST' j '_A'])));
64.     set(eval(['handles.edit_st' j '_saturation']), 'String',...
65.         num2str(evalin('base', ['ST' j '_sat'])));
66.     set(eval(['handles.edit_st' j '_ampFreq']), 'String',...
67.         num2str(evalin('base', ['ST' j '_Fa'])));
68.     set(eval(['handles.edit_st' j '_startFreq']), 'String',...
69.         num2str(evalin('base', ['ST' j '_F_start'])));
70.     set(eval(['handles.edit_st' j '_endFreq']), 'String',...
71.         num2str(evalin('base', ['ST' j '_F_end'])));
72.     set(eval(['handles.edit_st' j '_tsweep']), 'String',...
73.         num2str(evalin('base', ['ST' j '_F_tsweep'])));
74. end
75. % update handles structure
76. guidata(hObject, handles);
77.
78. % update GUI
79. updateFields(1, handles);
80. updateFields(2, handles);
81. updatePreview(1, handles)
82. updatePreview(2, handles)
```

Function *updatePreview*:

```
558. function updatePreview(stage, handles)
559. % This function uptates the signal preview for the stage given as input
560. % parameter in the signal genertaor GUI. All setting fields are taken
     into
561. % account. At the same time, this function updates the variables in the
562. % MATLAB workspace. Every time when a setting is changes, this function
     is
563. % called.
564. % The duration of the preview is the same as the simulation duration, the
565. % signal delay is not taken into account. If the simulation duration is
     set
566. % to 'inf' a maximum of 100s is displayed.
567.
568. % read input, transform to String
569. stStr = num2str(stage);
570.
571. % get simulation duration from GUI
572. tSimStr = get(handles.edit_simDuration, 'String');
573.
574. % set preview duration to 100s if simulation is set to 'inf'
575. if strcmp(tSimStr, 'inf')
576.     tSim = 100;
577. else
578.     tSim = str2double(tSimStr);
579. end
580.
581. % supdate the simulation duration in the workspace
582. assignin('base','SIM_DURATION',tSimStr);
583.
584. % time vector for the preview
585. t = 0:0.04:tSim;
586.
587. % init intermediate variales used to calculate the sine wave
588. A = 0;      % signal amplitude
589. sat = 1;    % saturation level
590. y1 = 1;     % function for amplitude sweep, default no sweep
591. y2 = 1;     % function for frequency sweep, default no sweep
592.
593. % calculate the generated signal based on input settings from the GUI
594. if get(eval(['handles.cb_st' stStr '_enable']), 'Value') == 1
595.     % stagae is enabled
596.
597.     % get amplitude from GUI
598.     A = str2double(get(eval(['handles.edit_st' stStr '_amp']),
     'String'));
599.     % get saturation from GUI
600.     sat = str2double(get(eval(['handles.edit_st' stStr
     '_saturation']),...
601.         'String'));
602.
603.     % update settings in the workspace
604.     assignin('base',['ST' stStr '_A'], A);
605.     assignin('base',['ST' stStr '_sat'], sat);
606.
607.     % check amplitude sweep settings from GUI
608.     if get(eval(['handles.cb_st' stStr '_ampSweep']), 'Value') == 1
609.         % amplitude sweep is enabled
610.
611.         % get amplitude sweep frequency from GUI
612.         fa = str2double(get(eval(['handles.edit_st' stStr
     '_ampFreq']),...
613.             'String'));
614.
615.         % update settings in the workspace
616.         assignin('base',['ST' stStr '_Fa'], fa);
```

```
617.         assignin('base',['ST' stStr '_A_SWEEP_EN'],1);
618.
619.         % calculate amplitude sweep function
620.         y1 = sin(2*pi*fa*t);
621.
622.     else
623.         % amplitude sweep is not enabled
624.
625.         % update setting in the workspace
626.         assignin('base',['ST' stStr '_A_SWEEP_EN'],0);
627.         y1 = 1;
628.     end
629.
630.     % check frequency sweep settings from GUI
631.     if get(eval(['handles.cb_st' stStr '_freqSweep']), 'Value') == 1
632.         % frequency sweep is enabled
633.
634.         % get frequency sweep parameter from GUI
635.         fstart = str2double(get(eval(['handles.edit_st' stStr...
636.             '_startFreq']), 'String'));
637.         fend = str2double(get(eval(['handles.edit_st' stStr
     '_endFreq'])...
638.             , 'String'));
639.         tsweep = str2double(get(eval(['handles.edit_st' stStr...
640.             '_tsweep']), 'String'));
641.
642.         % update settings in the workspace
643.         assignin('base',['ST' stStr '_F_start'], fstart);
644.         assignin('base',['ST' stStr '_F_end'], fend);
645.         assignin('base',['ST' stStr '_F_tsweep'], tsweep);
646.         assignin('base',['ST' stStr '_F_SWEEP_EN'], 1);
647.
648.         % calculate frequency sweep function
649.         y2 = chirp(t,fstart,tsweep,fend);
650.
651.     else
652.         % frequency sweep is not enabled
653.
654.         % update setting in the workspace
655.         assignin('base',['ST' stStr '_F_SWEEP_EN'], 0);
656.         y2 = 1;
657.     end
658. else
659.     % stage is not enabled
660.
661.     % update setting in the workspace
662.     assignin('base',['ST' stStr '_A'], A);
663. end
664.
665. % calculate output singnal based on ampltude, amplitude sweep and
     frequency
666. % sweep funcion
667. out = A.*y1.*y2;
668. % take saturation into account
669. out = min(sat, max(-sat, out));
670.
671. % select plot area of given stage
672. axes(eval(['handles.axes_st' stStr]));
673.
674. % plot signal preview
675. plot(t, out);
676. title(['Stage ' stStr ' Signal Preview']);
```

Function updateFields:

```
679. function updateFields(stage, handles)
680. % This function updates the access to the different fields in the signal
681. % generator GUI dependand on selected check box settintgs. For example if
682. % one stage is disabled, the user may not alter the settings for this
683. % stage.
684.
685. % read input, transform to String
686. stStr = num2str(stage);
687.
688. % get check box states
689. stEn = get(eval(['handles.cb_st' stStr '_enable']) , 'Value');
690. ampEn = get(eval(['handles.cb_st' stStr '_ampSweep']), 'Value');
691. freqEn = get(eval(['handles.cb_st' stStr '_freqSweep']), 'Value');
692.
693. % update check box states on workspace variables
694. assignin('base',['ST' stStr '_EN'], stEn);
695. assignin('base',['ST' stStr '_A_SWEEP_EN'], ampEn);
696. assignin('base',['ST' stStr '_F_SWEEP_EN'], freqEn);
697.
698. if stEn == 0
699.     % stage is disabled, deactivate all setting fields
700.
701.     % set ampllitude to 0
702.     assignin('base',['ST' stStr '_A'], 0);
703.
704.     set(eval(['handles.cb_st' stStr '_ampSweep']), 'enable', 'off');
705.     set(eval(['handles.cb_st' stStr '_freqSweep']), 'enable', 'off');
706.     set(eval(['handles.edit_st' stStr '_amp']), 'enable', 'off');
707.     set(eval(['handles.edit_st' stStr '_saturation']), 'enable', 'off');
708.     set(eval(['handles.edit_st' stStr '_ampFreq']), 'enable', 'off');
709.     set(eval(['handles.edit_st' stStr '_startFreq']), 'enable', 'off');
710.     set(eval(['handles.edit_st' stStr '_endFreq']), 'enable', 'off');
711.     set(eval(['handles.edit_st' stStr '_tsweep']), 'enable', 'off');
712. else
713.     % stage is enabled
714.
715.     % set ampllitude in the workspace
716.     assignin('base',['ST' stStr '_A'],...
717.         str2double(get(eval(['handles.edit_st' stStr '_amp']),
     'String')));
718.
719.     % activate stage specific settings
720.     set(eval(['handles.edit_st' stStr '_amp']), 'enable', 'on')
721.     set(eval(['handles.edit_st' stStr '_saturation']), 'enable', 'on');
722.     set(eval(['handles.cb_st' stStr '_ampSweep']), 'enable', 'on');
723.     set(eval(['handles.cb_st' stStr '_freqSweep']), 'enable', 'on');
724.
725.     if ampEn == 1
726.         % amplitude sweep is enabled
727.         set(eval(['handles.edit_st' stStr '_ampFreq']), 'enable', 'on');
728.     else
729.         % amplitude sweep is disabled
730.         set(eval(['handles.edit_st' stStr '_ampFreq']), 'enable', 'off');
731.     end
732.
733.     if freqEn == 1
734.         % frequency sweep is enabled
735.         set(eval(['handles.edit_st' stStr '_startFreq']), 'enable',
     'on');
736.         set(eval(['handles.edit_st' stStr '_endFreq']), 'enable', 'on');
737.         set(eval(['handles.edit_st' stStr '_tsweep']), 'enable', 'on');
738.     else
739.          % frequency sweep is disabled
740.         set(eval(['handles.edit_st' stStr '_startFreq']), 'enable',
     'off');
```

```
741.         set(eval(['handles.edit_st' stStr '_endFreq']), 'enable', 'off');
742.         set(eval(['handles.edit_st' stStr '_tsweep']), 'enable', 'off');
743.     end
744. end
```

9.31 Appendix 31: MATLAB code, *C_2DSFJ_Data_Aqu_main*

```
%% 2DSFJ Robot Data Aquisition Main Script
% This script hosts the basic settings to capture experiment data from
% the 2DSJF robot.

clc;
clear all;

%% Load Constants and Plant Specific Parameter (m-files):

constants
robot_2DSFJ_setup

%% Simulation Setup

SIM_NAME = 'C_2DSFJ_Data_Acq_SM';
DATA_DECIMATION = 25;       % log data every DATA_DECIAMTION * SIM_STEP s
SIM_DURATION = '60';        % sim duration (String), s
SIM_DELAY = 1;              % transport delay, s
                            % NOTE: the simulation duration in sec is:
                            % AQU_DURATION + AQU_DELAY
ST1_EN = 1;                 % enable 1 or disable 0 the stages
ST2_EN = 0;

%% Signal Generator Settings

% default settings:
% Stage 1:
ST1_A_SWEEP_EN = 1;         % enable the amplitude sweep
ST1_F_SWEEP_EN = 1;         % enable the frequency sweep

ST1_A = 0.08;               % amplitude, A
ST1_Fa = 0.1;               % frequency of the amplitude sweep, Hz

ST1_F_start = 1.5;          % frequency sweep start freq., Hz
ST1_F_end = 0.1;            % frequency sweep end freq., Hz
ST1_F_tsweep = 46;          % time in s to sweep from start to end freq.

ST1_sat = 0.5;              % out signal saturation threshold

% Stage 2:
ST2_A_SWEEP_EN = 1;         % enable the amplitude sweep
ST2_F_SWEEP_EN = 1;         % enable the frequency sweep

ST2_A = 0.2;                % amplitude, A
ST2_Fa = 0.1;               % frequency of the amplitude sweep, Hz

ST2_F_start = 1.5;          % frequency sweep start freq., Hz
ST2_F_end = 0.1;            % frequency sweep end freq., Hz
ST2_F_tsweep = 38;          % time in s to sweep from start to end freq.

ST2_sat = 0.5;              % output signal saturation threshold


%% Open Simulation Diagram
open(SIM_NAME);
set_param(SIM_NAME, 'StopTime', ...
    num2str(str2double(SIM_DURATION) + SIM_DELAY));
set_param(SIM_NAME, 'FixedStep', num2str(SIM_STEP));

%% Open signal generator GUI
SignalGeneratorGUI
```

9.32 Appendix 32: MATLAB code, *C_2DSFJ_Data_Aqu*

```
%% Build the Data Sets
% This script must be run after the data acquisition experiment to
% transform the captured raw data into iddata objects used for the
% estimation process. The actuation signal parameters are stored
% automatically in the data sets as information.
% NOTE: Do not alter signal generator settings between the experiment
% executaion and the running of this script.

clc;

%% Name the dataset
DATASET = 'demo_st1';       % this name is stored inside the dataset
object

% check if real plant experiment data is available
if exist([SIM_NAME '.mat'],'file')

    % load measured data from real plant experiment into workspace
    load([SIM_NAME '.mat']);

    % set time step of aquired data
    Ts = DATA_DECIMATION * SIM_STEP;

    % extract data, form simulation output, remove transport delay
    startPoint = SIM_DELAY/Ts + 1;

    % delete old data sets
    if exist('dataset1','var')
        clear dataset1;
    end
    if exist('dataset2', 'var')
        clear dataset2;
    end

    %% Stage 1 data set:
    if ST1_EN
        % input: motor current
        in1 = stage1data.signals.values(startPoint:end,1);
        % output: 2DSFJ states; theta1, theta2, theta1_dot, theta2_dot
        out1(:,1:4) = stage1data.signals.values(startPoint:end,2:5);

        % create dataset
        dataset1 = iddata(out1,in1,Ts);
        dataset1.Name = DATASET;
        dataset1.InputName = INPUT_NAME_2DSFJ;
        dataset1.OutputName = OUTPUT_NAME_2DSFJ;
        dataset1.InputUnit = INPUT_UNIT_2DSFJ;
        dataset1.OutputUnit = OUTPUT_UNIT_2DSFJ;
        dataset1.Tstart = 0;
        dataset1.TimeUnit = 'sec';
        dataset1.InterSample = {'foh'};
        dataset1.ExperimentName = 'stage1';

        notes = sprintf(['2DSFJ robot: ' SIM_NAME ', '...
            'stage1 data aquisition \n'...
            'simulation duration: ' num2str(SIM_DURATION) ' s \n']);

        if ST1_A_SWEEP_EN
            if ST1_F_SWEEP_EN
            % amplitude and frequency sweep enabled
                notes = sprintf([notes...
                'frequency sweep: \t ' num2str(ST1_F_start)...
                ' - ' num2str(ST1_F_end) ' Hz over '...
                num2str(ST1_F_tsweep) ' s \n'...
```

```
64.                 'amplitude sweep: \t ' num2str(ST1_A) '*sine(2*pi*'...
65.                 num2str(ST1_Fa) ') A \n' ]);
66.             else
67.             % only amplitude sweep enabled
68.                 notes = sprintf([notes 'amplitude sweep: \t '
    num2str(ST1_A)...
69.                 '*sine(2*pi*' num2str(ST1_Fa) ') A \n' ]);
70.             end
71.         elseif ST1_F_SWEEP_EN
72.         % only frequency sweep enabled
73.             notes = sprintf([notes 'frequency sweep: \t '...
74.             num2str(ST1_F_start) ' - ' num2str(ST1_F_end) ' Hz over '...
75.             num2str(ST1_F_tsweep) ' s \n' 'amplitude: \t\t\t '...
76.             num2str(ST1_A) ' A \n']);
77.         else
78.         % step function
79.             notes = sprintf([notes 'step responce: \t\t step at t = 0
    \n'...
80.             'amplitude: \t\t\t ' num2str(ST1_A) ' A \n']);
81.         end
82.
83.         if ST1_sat >= ST1_A
84.             % no saturation
85.             notes = sprintf([notes 'saturation: \t\t none\n']);
86.         else
87.             % saturation in effect
88.             notes = sprintf([notes 'saturation: \t\t +/- '
    num2str(ST1_sat)...
89.             ' rad\n']);
90.         end
91.             dataset1.Notes = {notes};
92.             disp(notes);
93.     end
94.
95.     %% Stage 2 - Z Axis data set:
96.     if ST2_EN
97.         % input: motor current
98.         in2 = stage2data.signals.values(startPoint:end,1);
99.         % output: 2DSFJ states; theta1, theta2, theta1_dot, theta2_dot
100.        out2(:,1:4) = stage2data.signals.values(startPoint:end,2:5);
101.
102.        % create dataset
103.        dataset2 = iddata(out2,in2,Ts);
104.        dataset2.Name = DATASET;
105.        dataset2.InputName = INPUT_NAME_2DSFJ;
106.        dataset2.OutputName = OUTPUT_NAME_2DSFJ;
107.        dataset2.InputUnit = INPUT_UNIT_2DSFJ;
108.        dataset2.OutputUnit = OUTPUT_UNIT_2DSFJ;
109.        dataset2.Tstart = 0;
110.        dataset2.TimeUnit = 'sec';
111.        dataset2.InterSample = {'foh'};
112.        dataset2.ExperimentName = 'stage2';
113.
114.        notes = sprintf(['2DSFJ robot: ' SIM_NAME ', '...
115.            'stage2 data aquisition \n'...
116.            'simulation duration: ' num2str(SIM_DURATION) ' s \n']);
117.
118.        if ST2_A_SWEEP_EN
119.            if ST2_F_SWEEP_EN
120.            % amplitude and frequency sweep enabled
121.                notes = sprintf([notes...
122.                'frequency sweep: \t ' num2str(ST2_F_start)...
123.                ' - ' num2str(ST2_F_end) ' Hz over '...
124.                num2str(ST2_F_tsweep) ' s \n'...
125.                'amplitude sweep: \t ' num2str(ST2_A) '*sine(2*pi*'...
126.                num2str(ST2_Fa) ') A \n' ]);
127.            else
```

```
128.             % only amplitude sweep enabled
129.                 notes = sprintf([notes 'amplitude sweep: \t '
    num2str(ST2_A)...
130.                 '*sine(2*pi*' num2str(ST2_Fa) ') A \n' ]);
131.             end
132.         elseif ST2_F_SWEEP_EN
133.         % only frequency sweep enabled
134.             notes = sprintf([notes 'frequency sweep: \t '...
135.             num2str(ST2_F_start) ' - ' num2str(ST2_F_end) ' Hz over '...
136.             num2str(ST2_F_tsweep) ' s \n' 'amplitude: \t\t\t '...
137.             num2str(ST2_A) ' A \n']);
138.         else
139.         % step function
140.             notes = sprintf([notes 'step responce: \t\t step at t = 0
    \n'...
141.             'amplitude: \t\t\t ' num2str(ST2_A) ' A \n']);
142.         end
143.
144.         if ST2_sat >= ST2_A
145.             % no saturation
146.             notes = sprintf([notes 'saturation: \t\t none\n']);
147.         else
148.             % saturation in effect
149.             notes = sprintf([notes 'saturation: \t\t +/- '
    num2str(ST2_sat)...
150.             ' rad\n']);
151.         end
152.             dataset2.Notes = {notes};
153.             disp(notes);
154.     end
155.
156.     % open System Identification Toolbox GUI to verify build data sets
157.     ident;
158.
159. end
```

9.33 Appendix 33: MATLAB code, *C_2DSFJ_Model_Est_main*

```
%% Model Estimation for the 2DSFJ Robot
% This script contains the algorithm to estimate the models. Grey box
% modelling is used, the model structure must be defined as a function.

clc;
clear all;
format compact;

%% Load Constants
constants
load C_2DSFJ_DataObjects;           % load data sets to workspace

%% Estimation Setup
% Basic Settings
EST_AXIS = '1';                     % specify axis/stage to be estimated
estData = st1DemoData;              % dataset for estimation process
verData = st1DemoData2;             % dataset for verification

% get No. of data points from estimation data set
NO_DATA_POINTS = size(estData.SamplingInstants);
NO_DATA_POINTS = NO_DATA_POINTS(1)-1;

% Advanced Settings
MAX_ITER = 60;                      % max No. of iterations for estimation
PARS_MIN = 0;                       % min value for each parameter
PARS_MAX = 60;                      % max value for each parameter
ITER_POINTS = NO_DATA_POINTS;       % No. of data points per iteration,
                                    % should be the No. of data points
                                    % between amplitude zero crossings.
                                    %  Set to size of dataset to do just
   one
                                    % iteration.

FUNC_NAME = 'C_2DSFJ_Model';        % function name for estimation loop
PARS_NBR = 6;                       % No. of parameters in model structure
ORDER = [4 1 4];                    % model order: ny nu nx (No. of...
                                    % outputs, inputs and states)

%% Load Data
INIT_PARAMETER = ['pars_2DSFJ_stage' EST_AXIS];


load(INIT_PARAMETER);               % load initial parameter to workspace



%% Create Linear Grey Box Model:

% start parameter, initial guess 'Ktg' 'Ks' 'J1' 'B1' 'J2' 'B2':
START_PARS = eval(INIT_PARAMETER);

grey2DSFJ = idnlgrey(FUNC_NAME, ORDER, START_PARS);
grey2DSFJ.InputName = INPUT_NAME_2DSFJ;
grey2DSFJ.InputUnit = INPUT_UNIT_2DSFJ;
grey2DSFJ.OutputName = OUTPUT_NAME_2DSFJ;
grey2DSFJ.OutputUnit = OUTPUT_UNIT_2DSFJ;
setpar(grey2DSFJ, 'Name', {'Ktg' 'Ks' 'J1' 'B1' 'J2' 'B2'});
setpar(grey2DSFJ, 'Minimum',PARS_MIN * ones(1,PARS_NBR));
setpar(grey2DSFJ, 'Maximum',PARS_MAX * ones(1,PARS_NBR));

% create model with initial parameters for comparison:
greyInit = grey2DSFJ;

% select parameters not to be estimated:
```

```
% grey2DSFJ.Parameters(1).Fixed = true;       % dont estimate Ktg
grey2DSFJ.Parameters(2).Fixed = true;       % dont estimate Ks
% grey2DSFJ.Parameters(3).Fixed = true;       % dont estimate J1
% grey2DSFJ.Parameters(4).Fixed = true;       % dont estimate B1
% grey2DSFJ.Parameters(5).Fixed = true;       % dont estimate J2
% grey2DSFJ.Parameters(6).Fixed = true;       % dont estimate B2


% log file initialisation
log = zeros((NO_DATA_POINTS/ITER_POINTS +1),PARS_NBR);
log(1,:) = START_PARS;

for i = 1:(NO_DATA_POINTS/ITER_POINTS)
    fprintf('iteration: %d\n',i)    % display iteration No.

    start = (i-1)*ITER_POINTS+1;
    stop = i*ITER_POINTS;

    % estimate parameters using prediction error estimate method:
    grey2DSFJ = pem(estData(1:stop), grey2DSFJ, 'Display','On',...
        'InitialStates','Zero', 'MaxIter', MAX_ITER);

    parsModel = grey2DSFJ.Parameters;
    for j = 1:PARS_NBR
        log(i+1,j) = parsModel(j).Value;
    end
end

% display model with estimated parameters:
present(grey2DSFJ);

% show estimation model fit with verification data:
figure(1);
compare(verData, greyInit, grey2DSFJ);
```

9.34 Appendix 34: MATLAB code, *C_2DSFJ_model*

```
1.  /* Include libraries */
2.  #include "mex.h"
3.  #include <math.h>
4.
5.  /* Specify the number of outputs here. */
6.  #define NY 4
7.
8.  /* State equations */
9.  void compute_dx(double *dx, double *x, double *u, double **p)
10. {
11.     /* Declaration of model parameters and intermediate variables: */
12.     double *Ktg, *Ks, *J1, *B1, *J2, *B2;
13.
14.     /* Retrieve model parameters: */
15.     Ktg     = p[0];    /* Motor torque constant                     */
16.     Ks      = p[1];    /* Robot torsional joint stiffness           */
17.     J1      = p[2];    /* Motor side, moment of inertia             */
18.     B1      = p[3];    /* Motor side, viscous damping coefficient   */
19.     J2      = p[4];    /* Arm side, moment of inertia               */
20.     B2      = p[5];    /* Arm side, viscous damoing coefficient     */
21.
22.
23.     /* State variables: */
24.     /* x[0]: Rotational position of the motor.                     */
25.     /* x[1]: Rotational position of the robot arm.                 */
26.     /* x[2]: Rotational velocity of the motor.                     */
27.     /* x[3]: Rotational velocity of the robot arm.                 */
28.
29.     dx[0] = x[2];
30.     dx[1] = x[3];
31.     dx[2] = 1/J1[0]*(-Ks[0]*(x[0]-x[1]) - B1[0]*x[2]  + Ktg[0]*u[0]);
32.     dx[3] = 1/J2[0]*(Ks[0]*(x[0]-x[1]) - B2[0]*x[3]);
33.
34. }
35.
36. /* Output equations */
37. void compute_y(double *y, double *x)
38. {
39.     /* output: all state variables */
40.     y[0] = x[0];
41.     y[1] = x[1];
42.     y[2] = x[2];
43.     y[3] = x[3];
44. }
45.
46.
47. /*---------------------------------------------------------------------
    *
48.    DO NOT MODIFY THE CODE BELOW UNLESS YOU NEED TO PASS ADDITIONAL
49.    INFORMATION TO COMPUTE_DX AND COMPUTE_Y
50.
51.    To add extra arguments to compute_dx and compute_y (e.g., size
52.    information), modify the definitions above and calls below.
53.  *---------------------------------------------------------------------
    */
54.
55. void mexFunction(int nlhs, mxArray *plhs[],
56.                  int nrhs, const mxArray *prhs[])
57. {
58.     /* Declaration of input and output arguments. */
59.     double *x, *u, **p, *dx, *y, *t;
60.     int     i, np, nu, nx;
61.     const mxArray *auxvar = NULL; /* Cell array of additional data. */
62.
```

```
63.     if (nrhs < 3) {
64.         mexErrMsgIdAndTxt("IDNLGREY:ODE_FILE:InvalidSyntax",
65.         "At least 3 inputs expected (t, u, x).");
66.     }
67.
68.     /* Determine if auxiliary variables were passed as last input.  */
69.     if ((nrhs > 3) && (mxIsCell(prhs[nrhs-1]))) {
70.         /* Auxiliary variables were passed as input. */
71.         auxvar = prhs[nrhs-1];
72.         np = nrhs - 4; /* Number of parameters (could be 0). */
73.     } else {
74.         /* Auxiliary variables were not passed. */
75.         np = nrhs - 3; /* Number of parameters. */
76.     }
77.
78.     /* Determine number of inputs and states. */
79.     nx = mxGetNumberOfElements(prhs[1]); /* Number of states. */
80.     nu = mxGetNumberOfElements(prhs[2]); /* Number of inputs. */
81.
82.     /* Obtain double data pointers from mxArrays. */
83.     t = mxGetPr(prhs[0]);  /* Current time value (scalar). */
84.     x = mxGetPr(prhs[1]);  /* States at time t. */
85.     u = mxGetPr(prhs[2]);  /* Inputs at time t. */
86.
87.     p = mxCalloc(np, sizeof(double*));
88.     for (i = 0; i < np; i++) {
89.         p[i] = mxGetPr(prhs[3+i]); /* Parameter arrays. */
90.     }
91.
92.     /* Create matrix for the return arguments. */
93.     plhs[0] = mxCreateDoubleMatrix(nx, 1, mxREAL);
94.     plhs[1] = mxCreateDoubleMatrix(NY, 1, mxREAL);
95.     dx      = mxGetPr(plhs[0]); /* State derivative values. */
96.     y       = mxGetPr(plhs[1]); /* Output values. */
97.
98.     /*
99.       Call the state and output update functions.
100.
101.           Note: You may also pass other inputs that you might need,
102.           such as number of states (nx) and number of parameters (np).
103.           You may also omit unused inputs (such as auxvar).
104.
105.           For example, you may want to use orders nx and nu, but not time
    (t)
106.           or auxiliary data (auxvar). You may write these functions as:
107.               compute_dx(dx, nx, nu, x, u, p);
108.               compute_y(y, nx, nu, x, u, p);
109.         */
110.
111.         /* Call function for state derivative update. */
112.         compute_dx(dx, x, u, p);
113.
114.         /* Call function for output update. */
115.         compute_y(y, x);
116.
117.         /* Clean up. */
118.         mxFree(p);
119.     }
```

9.35 Appendix 35: MATLAB code, *C_Omni_Est_main*

```
%% Model Estimation for the Phantom Omni Haptic Device
% This script contains the algorithm to estimate the models. Grey box
% modelling is used, the model structure must be defined as a function.

clc;
clear all;
format compact;

%% Load Constants
constants
load C_Omni_DataObjects;           % load data sets to workspace

%% Estimation Setup
% Basic Settings
EST_AXIS = 'y';                     % specify axis/stage to be estimated,
                                    % i.e. 'y' or 'z'
estData = yAxisDemoData;            % dataset for estimation process
verData = yAxisDemoData2;           % dataset for verification

% get No. of data points from estimation data set
NO_DATA_POINTS = size(estData.SamplingInstants);
NO_DATA_POINTS = NO_DATA_POINTS(1)-1;

% Advanced Settings
MAX_ITER = 60;                      % max No. of iterations for estimation
PARS_MIN = 0;                       % min value for each parameter
PARS_MAX = 10;                      % max value for each parameter
FIRST_POINTS = 200;                 % No. of data points for linear est.
ITER_POINTS = 200;                  % No. of data points per iteration,
                                    % should be the No. of data points
                                    % between amplitude zero crossings.
                                    % Set to size of dataset to do just
                                    % one iteration.

PARS_NBR = 5;                       % No. of pars in linear model
   structure
NL_PARS_NBR = 9;                    % No. of pars in nonlinear model
   struc.
FUNC_NAME_1 = 'C_Omni_model_2DOF_c';    % function name for first est.
FUNC_NAME_2 = 'C_Omni_nl_model_2DOF_c'; % function name for estimation
   loop
ORDER = [2 1 4];                    % model order: ny nu nx (No. of...
                                    % outputs, inputs and states)

%% Load Data
INIT_PARAMETER = ['pars_Omni_' EST_AXIS 'Axis'];

load(INIT_PARAMETER);               % load initial parameter to workspace

%% Create Linear Grey Box Model:

% start parameter, initial guess Jm, Ja, Bg, Bm, Kg:
START_PARS = eval(INIT_PARAMETER);

greyOMNI = idnlgrey(FUNC_NAME_1, ORDER, START_PARS);
greyOMNI.InputName = INPUT_NAME_OMNI;
greyOMNI.InputUnit = INPUT_UNIT_OMNI;
greyOMNI.OutputName = OUTPUT_NAME_OMNI;
greyOMNI.OutputUnit = OUTPUT_UNIT_OMNI;
setpar(greyOMNI, 'Name', {'Jm' 'Ja' 'Bg' 'Bm' 'Kg'});
setpar(greyOMNI, 'Minimum',PARS_MIN * ones(1, PARS_NBR));
setpar(greyOMNI, 'Maximum',PARS_MAX * ones(1, PARS_NBR));

% select parameters not to be estimated:
```

```
% greyOMNI.Parameters(1).Fixed = true;       % dont estimate Jm
% greyOMNI.Parameters(2).Fixed = true;       % dont estimate Ja
% greyOMNI.Parameters(3).Fixed = true;       % dont estimate Bg
% greyOMNI.Parameters(4).Fixed = true;       % dont estimate Bm
% greyOMNI.Parameters(5).Fixed = true;       % dont estimate Kg

% estimate linear parameters:

greyOMNI = pem(estData(1:FIRST_POINTS), greyOMNI, 'Display','On',...
    'InitialStates','Estimate', 'MaxIter', MAX_ITER);

% show linear estimation model fit:
figure(1);
compare(estData(1:FIRST_POINTS),greyOMNI);

% display estimated parameters:
present(greyOMNI);

% set estimated parameters for non-linear estimation
initModel = greyOMNI.Parameters;

%% Non-linear estimation process
% nonlinear estimation log file initialisation:
log = zeros((NO_DATA_POINTS/ITER_POINTS +1), NL_PARS_NBR);

pars = ones(1,NL_PARS_NBR);          % parameter vector initialisation
pars(1) = initModel(1).Value;        % Jm
pars(2) = initModel(2).Value;        % Ja
pars(3) = initModel(3).Value;        % Bg
pars(4) = initModel(5).Value;        % Kg
pars(5) = 1;                         % Fc
pars(6) = 0.01;                      % Fv
pars(7) = 8;                         % Fcs
pars(8) = 1;                         % alpha
pars(9) = 0.01;                      % beta

log(1,:) = pars;                     % first line of log file, start values

    % create non-linear grey model:
    greyOMNInl = idnlgrey(FUNC_NAME_2, ORDER, pars);
    greyOMNInl.InputName = INPUT_NAME_OMNI;
    greyOMNInl.InputUnit = INPUT_UNIT_OMNI;
    greyOMNInl.OutputName = OUTPUT_NAME_OMNI;
    greyOMNInl.OutputUnit = OUTPUT_UNIT_OMNI;
    setpar(greyOMNInl, 'Name', {'Jm' 'Ja' 'Da' 'Ka' 'Fc' 'Fv' 'Fcs'
'alpha'...
        'beta'});
    setpar(greyOMNInl, 'Minimum',PARS_MIN);
    setpar(greyOMNInl, 'Maximum',PARS_MAX);

    for i = 1:(NO_DATA_POINTS/ITER_POINTS)
        fprintf('iteration: %d\n',i)    % display iteration No.

        start = (i-1)*ITER_POINTS+1;
        stop = i*ITER_POINTS;

        % estimate parameters using prediction error estimate method:
        greyOMNInl = pem(estData(1:stop), greyOMNInl, 'Display','On',...
            'InitialStates','Estimate', 'MaxIter', MAX_ITER);

        % show current non-linear estimation model fit:
        figure(1);
        compare(estData(start:stop), greyOMNInl);

        parsModel = greyOMNInl.Parameters;
        for j = 1:NL_PARS_NBR
            log(i+1,j) = parsModel(j).Value;
```

```
128.         end
129.     end
130.
131.     % display model with estimated parameters:
132.     present(greyOMNInl);
133.
134.     % show estimation model fit with verification data:
135.     figure(1);
136.     compare(estData, greyOMNI, greyOMNInl);
137.     figure(2);
138.     compare(verData, greyOMNI, greyOMNInl);
```

9.36 Appendix 36: MATLAB code, *C_Omni_model_2DOF_c*

NOTE: The gateway function is the same as in Appendix 34 line 55ff, it is not mentioned here again.

```
8.  /* State equations */
9.  void compute_dx(double *dx, double *x, double *u, double **p)
10. {
11.     /* Declaration of model parameters and intermediate variables: */
12.     double *Jm, *Ja, *Bg, *Bm, *Kg;
13.
14.     /* Retrieve model parameters: */
15.     Jm      = p[0];    /* Motor moment of inertia                    */
16.     Ja      = p[1];    /* Robot arm moment of inertia                */
17.     Bg      = p[2];    /* Gear-box linear damping coefficient        */
18.     Bm      = p[3];    /* Gear-box stiffness                         */
19.     Kg      = p[4];    /* Coulomb friction coefficient               */
20.
21.
22.     /* State variables:*/
23.     /* x[0]: Rotational position of the motor. */
24.     /* x[1]: Rotational position of the robot arm. */
25.     /* x[2]: Rotational velocity of the motor. */
26.     /* x[3]: Rotational velocity of the robot arm. */
27.
28.     dx[0] = x[2];
29.     dx[1] = x[3];
30.     dx[2] = 1/Jm[0]*(-Kg[0]*(x[0]-x[1]) -(Bm[0] + Bg[0])*x[2] + Bg[0]*x[3]
    +u[0]);
31.     dx[3] = 1/Ja[0]*(Kg[0]*(x[0]-x[1]) +Bg[0]*(x[2] -x[3]));
32.
33. }
34.
35. /* Output equations */
36. void compute_y(double *y, double *x)
37. {
38.     /* y[0]: Rotational position of the robot arm. */
39.     /* y[1]: Rotational velocity of the robot arm. */
40.     y[0] = x[1];
41.     y[1] = x[3];
42. }
```

9.37 Appendix 37: MATLAB code, *C_Omni_nl_model_2DOF_c*

NOTE: The gateway function is the same as in Appendix 34 line 55ff, it is not mentioned here again.

```
8.  /* State equations */
9.  void compute_dx(double *dx, double *x, double *u, double **p)
10. {
11.     /* Declaration of model parameters and intermediate variables: */
12.     double *Jm, *Ja, *Bg, *Kg, *Fc, *Fv, *Fcs, *alpha, *beta;
13.     double T_f;  /* Intermediate variable. */
14.
15.     /* Retrieve model parameters: */
16.     Jm      = p[0];    /* Motor moment of inertia                   */
17.     Ja      = p[1];    /* Robot arm moment of inertia               */
18.     Bg      = p[2];    /* Gear-box linear damping coefficient       */
19.     Kg      = p[3];    /* Gear-box stiffness                        */
20.     Fc      = p[4];    /* Coulomb friction coefficient              */
21.     Fv      = p[5];    /* Viscous friction coefficient              */
22.     Fcs     = p[6];    /* Striebeck friction coefficient.           */
23.     alpha   = p[7];    /* Striebeck smoothness coefficient.         */
24.     beta    = p[8];    /* Friction smoothness coefficient.          */
25.
26.     /* State variables:*/
27.     /* x[0]: Rotational position of the motor. */
28.     /* x[1]: Rotational position of the robot arm. */
29.     /* x[2]: Rotational velocity of the motor. */
30.     /* x[3]: Rotational velocity of the robot arm. */
31.
32.     /* Determine intermediate variables: */
33.     /* T_f: non-linear friction torque. (NOTE: sech(x) = 1/cosh(x)!) */
34.     T_f = Fv[0]*x[2]+(Fc[0] +
    Fcs[0]/(cosh(alpha[0]*x[2])))*tanh(beta[0]*x[2]);
35.
36.     dx[0] = x[2];
37.     dx[1] = x[3];
38.     dx[2] = 1/Jm[0]*(-Kg[0]*(x[0]-x[1]) -Bg[0]*x[2] + Bg[0]*x[3] + u[0] -
    T_f);
39.     dx[3] = 1/Ja[0]*(Kg[0]*(x[0]-x[1]) +Bg[0]*(x[2] -x[3]));
40.
41. }
42.
43. /* Output equations */
44. void compute_y(double *y, double *x)
45. {
46.     /* y[0]: Rotational position of the robot arm. */
47.     /* y[1]: Rotational velocity of the robot arm. */
48.     y[0] = x[1];
49.     y[1] = x[3];
50. }
```

9.38 Appendix 38: MATLAB code, *AB_Scen1_main*

```
%% Scenario 1
% This script hosts the basic settings to run the scenario 1 real plant
% exeriment or the simulaton. Choose the desired diagram by
% comment/uncomment the lines 22/23.
% Master:   Phantom Omni
% Slave:    2DSFJ robot

clc;
clear all;

%% Load Constants and Plant Specific Parameter (m-files):

constants;
robot_2DSFJ_setup;          % load plant specific parameter
phantomOmni_setup;
load C_2DSFJ_ModelObjects;  % load models


%% Simulation Setup

% experiment selection:
% SIM_NAME = 'A_Scen1_RealPlant';
SIM_NAME = 'B_Scen1_Sim';

DATA_DECIMATION = 25;       % log data every DATA_DECIAMTION * SIM_STEP s
SIM_DURATION = '9';         % sim duration (String), s
SIM_DELAY = 1;              % transport delay, s
                            % NOTE: the effective simulation duration in
                            % sec is: SIM_DURATION + SIM_DELAY

ST1_EN = 1;                 % enable 1 or disable 0 the stages
ST2_EN = 0;
FB_EN = 0;                  % feedback loop: 1 to enable, 0 to disable

% manual mode enable: 1 = manual control with Omni, 0 = torque simulated
% with signal generator
MAN_MODE_ST1 = 0;
MAN_MODE_ST2 = 0;

% fictitious Spring constant, used to maintain neutral position
Kfs = 1.7;                  % Nm/rad

Ky = 10;                     % applied human torque gains, Nm/rad
Kz = 10;

ST1_WALL = 1;
ST2_WALL = 1;

ST1_Model = st1DemoModel;   % select used model structure
ST2_Model = st2DemoModel;

%% Controller parameter:
% torque error gain constants
ST1_lambda = 0.5;
ST2_lambda = 0.5;

% stage 1 controller values:
ST1_Kp_1    = 0;
ST1_Ki      = 20;
ST1_Kp_2    = 0.25;
ST1_Kd      = 0.1;

% stage 2 controller Values:
ST2_Kp_1    = 0;
```

```
65.  ST2_Ki      = 20;
66.  ST2_Kp_2    = 0.25;
67.  ST2_Kd      = 0.1;
68.
69.  %% Signal Generator Settings
70.
71.  % default settings:
72.  % Stage 1:
73.  ST1_A_SWEEP_EN = 0;          % enable the amplitude sweep
74.  ST1_F_SWEEP_EN = 0;          % enable the frequency sweep
75.
76.  ST1_A = 0.1;                % amplitude, A
77.  ST1_Fa = 0.1;                % frequency of the amplitude sweep, Hz
78.
79.  ST1_F_start = 1.5;           % frequency sweep start freq., Hz
80.  ST1_F_end = 0.1;             % frequency sweep end freq., Hz
81.  ST1_F_tsweep = 46;           % time in s to sweep from start to end freq.
82.
83.  ST1_sat = 0.5;               % out signal saturation threshold
84.
85.  % Stage 2:
86.  ST2_A_SWEEP_EN = 0;          % enable the amplitude sweep
87.  ST2_F_SWEEP_EN = 0;          % enable the frequency sweep
88.
89.  ST2_A = 0.1;                 % amplitude, A
90.  ST2_Fa = 0.1;                % frequency of the amplitude sweep, Hz
91.
92.  ST2_F_start = 1.5;           % frequency sweep start freq., Hz
93.  ST2_F_end = 0.1;             % frequency sweep end freq., Hz
94.  ST2_F_tsweep = 38;           % time in s to sweep from start to end freq.
95.
96.  ST2_sat = 0.5;               % output signal saturation threshold
97.
98.  %% Feedback settings:
99.
100. FEEDBACK_GAIN = 0.6;         % feedback gain value
101. ST1_FB_FILTER = 0;           % feedback signal filter:
102. ST2_FB_FILTER = 0;           % 1 to enable, 0 to disable
103. % Feedback Filter Parameter:
104. wcfb = 2 * pi * 4;           % feedback signal cutting frequency (rad/s)
105.
106. %% Open Simulation Diagram
107. open(SIM_NAME);
108. set_param(SIM_NAME, 'StopTime', ....
109.     num2str(str2double(SIM_DURATION) + SIM_DELAY));
110. set_param(SIM_NAME, 'FixedStep', num2str(SIM_STEP));
111.
112. %% Open signal generator GUI,
113. if ~MAN_MODE_ST1 || ~MAN_MODE_ST1 || strcmp(SIM_NAME, 'B_Scen1_Sim')
114.     % if manual mode is disabled or the simulation diagram is selected
115.     SignalGeneratorGUI
116. end
```

9.39 Appendix 39: MATLAB code, *D_Scen1_Exp_Analysis*

```
%% Scenario 1 Experiment analysis
% Run this script after the simulaton or a real plant experiment to plot
% the captured data.

if strcmp(SIM_NAME, 'A_Scen1_RealPlant')
    % selected diagram is thee real plant

    name1 = 'Stage 1 Real Plant';
    name2 = 'Stage 2 Real Plant';

    if exist('A_Scen1_RealPlant.mat','file')
        % experiment data is available
        load A_Scen1_RealPlant.mat
    else
        disp('No experiment data found')
        return;
    end
else
    name1 = 'Stage 1 Simulation';
    name2 = 'Stage 2 Real Plant';
end

% set time step of aquired data
Ts = DATA_DECIMATION * SIM_STEP;

%% extract data form simulation output

% startPoint = SIM_DELAY/Ts + 1;          % remove transport delay:
startPoint = 1;                          % dont remove transport delay

if ST1_EN
% stage 1 was enabled
    % controller:
    clear control1 position1;
    control1(:,[1 2 3 4]) = controller1.signals.values(startPoint:end,...
        [1 2 3 4]);
    position1(:,1) = stage1data.signals.values(startPoint:end, 2);

    stopTime = size(control1);          % get no. of data points
    stopTime = (stopTime(1)-1) * Ts;    % convert to time span

    t =  0:Ts:stopTime;                 % time vector

    Scen1_createfigure(t, control1, position1, name1);
end

if ST2_EN
% stage 2 was enabled
    % controller:
    clear control2 position2;
    control2(:,[1 2 3 4]) = controller2.signals.values(startPoint:end,...
        [1 2 3 4]);
    position2(:,1) = stage2data.signals.values(startPoint:end, 2);

    stopTime = size(control2);          % get no. of data points
    stopTime = (stopTime(1)-1) * Ts;    % convert to time span

    t =  0:Ts:stopTime;                 % time vector

    Scen1_createfigure(t, control2, position2, name2);
end
```

9.40 Appendix 40: MATLAB code, *AB_Scen2_main*

```
%% Scenario 2
% This script hosts the basic settings to run the scenario 2 real plant
% exeriment or the simulaton. Choose the desired diagram by
% comment/uncomment the lines 23/24.
% Master:   2DSFJ robot
% Slave:    Phantom Omni

clc;
clear all;

%% Load Constants and Plant Specific Parameter (m-files):

constants;
robot_2DSFJ_setup;          % load plant specific parameter
phantomOmni_setup;

load C_2DSFJ_ModelObjects;  % load models
load C_Omni_ModelObjects;

%% Simulation Setup

% experiment selection:
% SIM_NAME = 'A_Scen2_RealPlant';
SIM_NAME = 'B_Scen2_Sim';

DATA_DECIMATION = 25;       % log data every DATA_DECIAMTION * SIM_STEP s
SIM_DURATION = '10';        % sim duration (String), s. 'inf' for
infinite
SIM_DELAY = 1;              % transport delay, s
                            % NOTE: the simulation duration in sec is:
                            % SIM_DURATION + SIM_DELAY

ST1_EN = 1;                 % enable 1 or disable 0 the stages
ST2_EN = 0;
FB_EN = 1;                  % feedback loop: 1 to enable, 0 to disable
                            % the position reflection to the master.

% manual mode enable: 1 = manual control with 2DSFJ, 0 = robot arm
position
% simulated with signal generator.
MAN_MODE_ST1 = 0;
MAN_MODE_ST2 = 0;

yAXIS_WALL = 1;             % position in rad were the slave touches a
zAXIS_WALL = 0.5;           % virtual wall. Absolute value

ST1_Model = st1DemoModel;   % select used model structure
ST2_Model = st2DemoModel;

yAXIS_Model = yAxisDemoModel;
zAXIS_Model = zAxisDemoModel;
%% Controller Parameter

% Y axis:
Y_Ks = Ks1;                 % torsional stiffness stage 1
Y_Kp1 = 0.1;                % controller gain value
Y_Lambda = 0.5;             % torque gain factor

% Z axis:
Z_Ks = Ks2;                 % torsional stiffness stage 2
Z_Kp1 = 0.05;               % controller gain value
Z_Lambda = 0.3;             % torque gain factor

%% Reflection Controller Parameter:
```

```
63.  % Stage 1
64.  ST1_KPreflect = 5;
65.  ST1_KIreflect = 5;
66.  ST1_KDreflect = 0.1;
67.
68.  % Stage 2
69.  ST2_KPreflect = 5;
70.  ST2_KIreflect = 5;
71.  ST2_KDreflect = 0;
72.
73.  %% Signal Generator Settings
74.
75.  % default settings:
76.  % Stage 1:
77.  ST1_A_SWEEP_EN = 1;          % enable the amplitude sweep
78.  ST1_F_SWEEP_EN = 0;          % enable the frequency sweep
79.
80.  ST1_A = 0.5;                 % amplitude, rad
81.  ST1_Fa = 0.2;                % frequency of the amplitude sweep, Hz
82.
83.  ST1_F_start = 1.5;           % frequency sweep start freq., Hz
84.  ST1_F_end = 0.1;             % frequency sweep end freq., Hz
85.  ST1_F_tsweep = 46;           % time in s to sweep from start to end freq.
86.
87.  ST1_sat = 0.5;               % out signal saturation threshold
88.
89.  % Stage 2:
90.  ST2_A_SWEEP_EN = 0;          % enable the amplitude sweep
91.  ST2_F_SWEEP_EN = 0;          % enable the frequency sweep
92.
93.  ST2_A = 0.2;                 % amplitude, A
94.  ST2_Fa = 0.1;                % frequency of the amplitude sweep, Hz
95.
96.  ST2_F_start = 1.5;           % frequency sweep start freq., Hz
97.  ST2_F_end = 0.1;             % frequency sweep end freq., Hz
98.  ST2_F_tsweep = 38;           % time in s to sweep from start to end freq.
99.
100. ST2_sat = 0.5;               % output signal saturation threshold
101.
102. %% Open Simulation Diagram
103. open(SIM_NAME);
104. set_param(SIM_NAME, 'StopTime',....
105.     num2str(str2double(SIM_DURATION) + SIM_DELAY));
106. set_param(SIM_NAME, 'FixedStep', num2str(SIM_STEP));
107.
108. %% Open signal generator GUI
109. if ~MAN_MODE_ST1 || ~MAN_MODE_ST1 || strcmp(SIM_NAME, 'B_Scen1_Sim')
110.     % if manual mode is disabled or the simulation diagram is selected
111.     SignalGeneratorGUI
112. end
```

9.41 Appendix 41: MATLAB code, *D_Scen1_Exp_Analysis*

```
%% Scenario 2 Experiment analysis
% Run this script after the simulaton or a real plant experiment to plot
% the captured data.

if strcmp(SIM_NAME, 'A_Scen2_RealPlant')
    % selected diagram is thee real plant

    name1 = 'Y-Axis Real Plant';
    name2 = 'Z-Axis Real Plant';

    if exist('A_Scen2_RealPlant.mat','file')
         % experiment data is available
        load A_Scen2_RealPlant.mat
    else
        disp('No experiment data found');
        return;
    end

else
    name1 = 'Y-Axis Simulation';
    name2 = 'Z-Axis Simulation';
end

% set time step of aquired data
Ts = DATA_DECIMATION * SIM_STEP;

%% extract data form simulation output

startPoint = SIM_DELAY/Ts + 1;          % remove transport delay:
% startPoint = 1;                         % dont remove transport delay

if ST1_EN
% stage 1 / Y-axis was enabled

    clear position1 reflect1 velocity1 torque1;

    % position: master and slave arm position
    position1(:,1) = stage1data.signals.values(startPoint:end,2);
    position1(:,2) = omniData.signals.values(startPoint:end,1);
    % reflect: slave arm and master motor position
    reflect1(:,1) = omniData.signals.values(startPoint:end,1);
    reflect1(:,2) = stage1data.signals.values(startPoint:end,1);
    % velocity: master and slave arm velocity
    velocity1(:,1) = stage1data.signals.values(startPoint:end,4);
    velocity1(:,2) = omniData.signals.values(startPoint:end,3);
    % torque: desited torque and applied torque
    torque1(:,1) = Ks1 * ...
        (stage1data.signals.values(startPoint:end,2) -...
        stage1data.signals.values(startPoint:end,1));
    torque1(:,2) = controller1.signals.values(startPoint:end,3);

    stopTime = size(position1);         % get no. of data points
    stopTime = (stopTime(1)-1) * Ts;    % convert to time span

    t =  0:Ts:stopTime;                 % time vector

    Scen2_createfigure(t, position1, reflect1, velocity1, torque1, name1);
end

if ST2_EN
% stage 2 / Z-axis was enabled

    clear position2 reflect2 velocity2 torque2;

```

```
65.     % position: master and slave arm position
66.     position2(:,1) = stage2data.signals.values(startPoint:end,2);
67.     position2(:,2) = omniData.signals.values(startPoint:end,2);
68.     % reflect: slave arm and master motor position
69.     reflect2(:,1) = omniData.signals.values(startPoint:end,2);
70.     reflect2(:,2) = stage2data.signals.values(startPoint:end,1);
71.     % velocity: master and slave arm velocity
72.     velocity2(:,1) = stage2data.signals.values(startPoint:end,4);
73.     velocity2(:,2) = omniData.signals.values(startPoint:end,4);
74.     % torque: desited torque and applied torque
75.     torque2(:,1) = Ks1 * ...
76.         (stage2data.signals.values(startPoint:end,2) -...
77.         stage2data.signals.values(startPoint:end,1));
78.     torque2(:,2) = controller2.signals.values(startPoint:end,3);
79.
80.     stopTime = size(position2);          % get no. of data points
81.     stopTime = (stopTime(1)-1) * Ts;     % convert to time span
82.
83.     t =  0:Ts:stopTime;                  % time vector
84.
85.     Scen2_createfigure(t, position2, reflect2, velocity2, torque2, name2);
86. end
```